Available online at www.sciencedirect.com

SCIENCE DIRECT®

Discrete Applied Mathematics 154 (2006) 2212–2238

DISCRETE APPLIED MATHEMATICS

www.elsevier.com/locate/dam

A stabilized column generation scheme for the traveling salesman subtour problem

Andreas Westerlund*, Maud Göthe-Lundgren, Torbjörn Larsson

Department of Mathematics, Linköping University, SE-581 83 Linköping, Sweden

Received 10 September 2002; received in revised form 29 June 2004; accepted 21 April 2005
Available online 29 March 2006

Abstract

Given an undirected graph with edge costs and both revenues and weights on the vertices, the traveling salesman subtour problem is to find a subtour that includes a depot vertex, satisfies a knapsack constraint on the vertex weights, and that minimizes edge costs minus vertex revenues along the subtour.

We propose a decomposition scheme for this problem. It is inspired by the classic side-constrained 1-tree formulation of the traveling salesman problem, and uses stabilized column generation for the solution of the linear programming relaxation. Further, this decomposition procedure is combined with the addition of variable upper bound (VUB) constraints, which improves the linear programming bound. Furthermore, we present a heuristic procedure for finding feasible subtours from solutions to the column generation problems. An extensive experimental analysis of the behavior of the computational scheme is presented.

MSC: 90C27; 90B10

Keywords: Traveling salesman subtour problem; Prize collecting traveling salesman problem; Stabilized column generation; 1-tree problem with one degree-constraint

1. Introduction

Consider a complete and undirected graph $G = (V, E)$ with edge costs w_e, $e \in E$, that satisfy the triangle inequality. To every vertex $v \in V$ is associated a vertex revenue, r_v, and a vertex weight, d_v, which is non-negative. A specific vertex, v_1, corresponds to a depot, for which $r_{v_1} = d_{v_1} = 0$. The *traveling salesman subtour problem* (TSSP) amounts to identifying a cycle (tour, route) in the graph that visits a set of vertices, which includes v_1, that satisfies a knapsack restriction $\sum_{v \in T} d_v \leqslant D$, where T denotes the vertices covered by the cycle and D is a positive constant, and that minimizes travel cost minus vertex revenues along the cycle.

Choosing $d_v = 0$ and $r_v = +\infty$ for all $v \in V \backslash \{v_1\}$, T will be equal to V in an optimal solution for the problem, which is then obtained by solving a traveling salesman problem (TSP). This observation shows that TSP, which is an $\mathcal{NP}$-hard problem, is a special case of the traveling salesman subtour problem, which is therefore also $\mathcal{NP}$-hard.

The traveling salesman subtour problem is only one of many generalizations of TSP where it is not necessary to visit all vertices. A unifying framework for this type of problems, which is called the single vehicle routing–allocation

* Corresponding author.
E-mail address: anwes@mai.liu.se (A. Westerlund).

0166-218X/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.dam.2005.04.012

problem (SVRAP), is presented in [9]. In a solution to a SVRAP, a vertex can belong to one of three categories: it can be *on* the tour (i.e. visited by the tour), *off* the tour but directly allocated to a vertex on the tour, or the vertex can be left *isolated* (that is, neither on the tour nor allocated to a vertex that is on the tour). Allocating vertex v_i off the tour to vertex v_j on the tour, has a cost $h_{i,j}$, and leaving vertex v isolated has an associated cost h_v. In certain versions of SVRAP there is also some type of knapsack constraint involved, that puts a limit on the number of vertices on the tour. Moreover, some vertices can be required to be on the tour, off tour or isolated. Often a specific vertex, corresponding to a depot, must be on the tour.

In [9], it is shown that a number of optimization problems are special cases or slight modifications of SVRAP. Some of these problems are the *covering salesman problem*, the *median cycle problem*, the *maximal covering tour problem*, the traveling salesman problem, the *generalized traveling salesman problem*, the *selective traveling salesman problem* and the *prize collecting traveling salesman problem*. As an example, the median cycle problem (see [35]) is obtained as a special case of SVRAP when no vertex is allowed to be isolated (i.e. if $h_v := +\infty$, $v \in V$).

An important class of special cases of SVRAP is where no allocation of off-tour vertices to on-tour vertices is possible [i.e. if $h_{i,j} := +\infty, \forall(v_i, v_j)$]. This means that a vertex is either on the tour or isolated. A survey of this category of problems is presented in [17]. In their context, there is a profit (revenue) associated with each vertex, and they give an overview of traveling salesman problems with profits. They subdivide these optimization problems into three subclasses:

1. If the tour shall minimize travel cost, and the collected profit must not be smaller than a given value $p_{\min}$, the problem belongs to the class of the Quota traveling salesman problem (Quota TSP). This class is clearly a special case of SVRAP.
2. If the tour shall maximize collected profit, and the travel cost must not exceed a given value $c_{\max}$, the problem belongs to the class of the selective traveling salesman problem (Selective TSP). In this problem, a subset $\Gamma \subset V$ of the vertices is required to be on the route, and it is a special case of SVRAP where the travel cost of the route is moved from the objective function into a knapsack constraint. Furthermore, instead of maximizing the collected profit, one can minimize the sum of profits associated with the isolated vertices.
3. If both travel cost and profit are combined in the objective function, a profitable tour problem (PTP) is obtained. Normally, the objective function is to minimize travel cost minus profit. To see that PTP is a special case of SVRAP, put $h_v := 0$, $v \in V$, and subtract vertex profits from the edge costs according to $c'_{i,j} := c_{i,j} - r_{v_i}/2 - r_{v_j}/2$, and find a minimum cost tour with respect to the costs $c'_{i,j}$. This transformation works, because each vertex on a tour has two incident edges in a solution.
   Often, a knapsack constraint is added to PTP. The generic name is then the traveling salesman subset-tour problem with one additional constraint (TSSP+1). The traveling salesman subtour problem, which is the topic of this paper, belongs to this class of problems.

An approximation algorithm for a problem that belongs to the Quota TSP class is presented in [2].

For a survey of problems in the class of the Selective TSP, see [16]. A special case of the Selective TSP is obtained if the set of compulsory vertices consists of only the depot, i.e. if $\Gamma = \{v_1\}$. This case is known as the *orienteering problem*, or the *score orienteering problem*. Heuristics for this problem are given in [50,24,25,48], and exact methods are presented in [32,37,49,18,21]. In [37] a branch-and-bound method is developed. Depending on problem characteristics, instances including between 10 and 90 vertices were solved to optimality. In [32], a branch-and-cut algorithm is proposed, and it can solve to optimality instances involving up to 500 nodes. Another branch-and-cut method is given in [21]. It can solve certain instances involving up to 300 vertices.

A linear programming relaxation approach is given in [40]. In [33], a directed 1-subtree relaxation of the orienteering problem is used. They note that their 1-subtree problem is $\mathcal{NP}$-hard and propose a combination of cutting planes and Lagrangian relaxation for its solution.

For the profitable tour problem, [54] proposes a graph transformation that turns this problem into a standard asymmetric TSP. This transformation was further utilized in [14]. In [34,7], valid inequalities and facets for the PTP polytope are studied. The profitable tour problem has also been approached by approximation algorithms, see [23,10].

As stated above, an important case of the profitable tour problem is when a knapsack constraint is appended, i.e. problems that belong to the class TSSP+1. Perhaps the most well known problem of this type is the prize collecting traveling salesman problem (PCTSP). It is the problem of minimizing travel cost along the cycle plus total vertex

penalty for isolated vertices, subject to that the total vertex revenue along the cycle must be greater or equal to a specified amount. The PCTSP was introduced in [5] as a model for scheduling the daily operation of a steel rolling mill. Studies on PCTSP include structural properties of its polytope, see [3,4]. Bounding procedures based on relaxations have been developed in [19,14]. In [19], instances with up to 100 nodes are solved. In [13], a Lagrangian heuristic for the PCTSP is presented. Another type of TSSP+1 is studied in [22]. In this problem, two types of data are associated with an edge: travel cost and travel time. The objective is to minimize total travel costs minus profits subject to a restriction on the total time spent along the cycle. Branch-and-bound is made with bounding based on a relaxation of the problem into time-constrained assignment problems.

In [43] an insert/delete heuristic is presented for a TSSP+1 problem where the knapsack constraint is expressed on edges. This problem is further studied in the thesis [47],where a polyhedral analysis is performed and a branch-and-cut method is developed. In [45], this problem occurs as a subproblem in a decomposition approach for the solution of the vehicle routing problem. This work is continued in [12], where a parallel implementation is performed. They use one processor for each vehicle, and apply the insert/delete heuristic on each vehicle in parallel.

Finally, the traveling salesmen subtour problem, introduced in the beginning of this section, is a TSSP+1 problem where the knapsack constraint is expressed on vertices. This constraint can be thought of as a vehicle capacity, for a vehicle that is used to serve customers on a route. This optimization problem is studied in [27], where some alternative Lagrangian relaxation techniques are studied. In [26], this problem appears as a subproblem in a solution method for the basic vehicle routing game, and it is there solved by an adapted version of the procedure in [37]. The computational tests indicate that the algorithm rapidly becomes computationally burdensome when the number of vertices increases. Moreover, the traveling salesmen subtour problem is studied in [11], under the name the *capacitated prize-collecting traveling salesman problem*. A branch-and-cut solution method is proposed and used to solve instances with between 50 and 280 nodes.

In this paper, we present a stabilized column generation method for the solution of the traveling salesman subtour problem. The aim is to generate lower bounds to its optimal value, as well as integer feasible solutions and upper bounds. In order to find feasible solutions, we exploit the fact that it is relatively easy to recover overall feasibility from a solution to the column generation subproblem. The simplicity of finding integer feasible solutions is in contrast to the situation in solution methods based on linear programming relaxations, where it is typically difficult to recover integer feasible solutions from fractional solutions. (The reader may compare our strategy to that of Lagrangian heuristics, where feasible solutions are recovered from solutions to relaxed problems.)

The outline of the paper is as follows. In Section 2, a formulation and decomposition of the problem is derived. The stabilized column generation scheme is presented in Section 3, and in Section 3.4 a method for late inclusion of a small set of bound-improving variable upper bound constraints is given. In Section 4 a method for finding integer feasible solutions is stated, and in Section 5 computational results are given. Concluding remarks are made in Section 6.

A contribution of our work is that we propose a decomposition of the traveling salesman subtour problem, which takes advantage of a substructure that resembles the classical side-constrained 1-tree formulation of TSP, see [28]; we exploit this structure in a column generation scheme. Another contribution is that we establish the importance of stabilizing the column generation scheme. Further, we show that the addition of variable upper bound constraints to the master program of the column generation scheme improves the quality of the lower bounds as well as the quality of the integer solutions found by a feasibility heuristic. However, the addition of such constraints significantly slows down the practical rate of convergence of the algorithm.

2. Preliminaries

In this section some alternative mathematical formulations of the traveling salesman subtour problem are given. The purpose is to derive a formulation that is amenable to mathematical decomposition.

2.1. The model and its derivation

Consider the traveling salesman subtour problem (TSSP) introduced in the beginning of Section 1, where $T \subseteq V$ is the set of vertices visited by the cycle. A feasible solution to this optimization problem belongs to one of three categories: $|T| = 1$, $|T| = 2$ or $|T| \geqslant 3$. The first two categories correspond to $|V|$ solutions, namely the empty cycle and

$|V|-1$ cardinality two cycles, all of which can easily be enumerated, and the best of these solutions has the objective value $\Pi = \min\{0, \min_{v\in V\setminus\{v_1\}}(2c_{v_1,v} - r_v)\}$. Therefore, in the remainder of this paper, we consider the third category only and denote this problem by TSSP_3. This problem is assumed to have a feasible solution (which is the case if the sum of the three smallest vertex weights is not larger than D). Naturally, an optimal solution to TSSP_3 is of interest only if its objective value is smaller than Π.

We assume that G is a complete graph and that the cost matrix satisfies the triangle inequality. The latter assumption makes sure that each edge is used at most once in an optimal solution to TSSP_3. Given any $S \subseteq V$, let $E(S)$ denote the set of edges with both end-vertices in S and let $\delta(S)$ denote the set of edges with one end-vertex in S and the other in $V\setminus S$. When $S = \{v\}$, we write $\delta(v)$ rather than $\delta(\{v\})$, for brevity. Let

$$x_e = \begin{cases} 1 & \text{if edge } e \text{ is included in the cycle,} \\ 0 & \text{otherwise,} \end{cases} \qquad e \in E,$$

and

$$y_v = \begin{cases} 1 & \text{if vertex } v \text{ is included in the cycle,} \\ 0 & \text{otherwise.} \end{cases} \qquad v \in V,$$

For $J \subseteq E$ and $S \subseteq V$, define $x(J) = \sum_{e\in J} x_e$ and $y(S) = \sum_{v\in S} y_v$. Moreover, define $\mathcal{S} = \{S \subset V : (|S| \geqslant 2, v_1 \notin S) \vee (|S| = 2, v_1 \in S)\}$. A straightforward linear integer programming formulation of TSSP_3 then is

$$\begin{aligned}
[P] \quad \min \quad & w^{\mathrm{T}}x - r^{\mathrm{T}}y \\
\text{s.t.} \quad & x(\delta(v)) = 2y_v, \quad v \in V, && \text{(1a)}\\
& x(E(S)) \leqslant y(S) - y_{v_k}, \quad S \subseteq \mathcal{S}, \ v_k \in S, && \text{(1b)}\\
& d^{\mathrm{T}}y \leqslant D, && \text{(1c)}\\
& y_{v_1} = 1, && \text{(1d)}\\
& x_e \in \{0,1\}, \quad e \in E, && \text{(1e)}\\
& y_v \in \{0,1\}, \quad v \in V. && \text{(1f)}
\end{aligned}$$

Constraints (1a) are vertex degree restrictions. The restrictions (1b) are generalized subtour elimination constraints, see [3]. Normally, there are no such constraints for subsets $S \ni v_1$, since such constraints would make the set of feasible solutions empty. However, since we have assumed that all solutions pass through at least three vertices, it is possible to use the constraints also when $v_1 \in S$ and $|S| = 2$. The knapsack constraint (1c) limits the set of vertices visited by the cycle. The restriction (1d) ensures that the depot is included in the cycle.

In the remainder of this section we explore the connections between generalized subtour elimination constraints and connectivity constraints. Furthermore, a set of redundant constraints is appended to the model. The purpose of this is to obtain a convenient subproblem in a decomposition scheme. This subproblem which, after a simple variable substitution, becomes similar to the well-known 1-tree problem introduced in [28], is discussed in detail in Section 3.3.

Lemma 1. *Let $S \subseteq V$ be nonempty and assume that the vertex degree constraints* (1a) *are satisfied. Then*, $2x(E(S)) + x(\delta(S)) = 2y(S)$.

Proof. This result is given in [36], and it is proved by summing up the constraints (1a) over all $v \in S$. □

Next, consider the constraint

$$x(E) = y(V), \tag{2a}$$

and the connectivity constraints

$$x(\delta(S)) \geqslant 1 + y(S) - |S|, \quad S \subseteq V\setminus\{v_1\}, \ |S| \geqslant 1. \tag{2b}$$

Choosing $S = V$ in Lemma 1 gives (2a); hence this constraint is redundant in P. Furthermore, if the constraint (2a) is introduced in a formulation of TSSP_3, one of the vertex degree restrictions (1a) can be dropped without changing the set of feasible solutions.

Let P_1 denote the model obtained from P by appending the constraint (2a), dropping the vertex degree restriction for vertex v_2, and appending the constraint (2b). The constraints in (2b) are, in fact, redundant in P_1, as shown below. The purpose of introducing (2a) and (2b) is, however, to create a convenient column generation problem, as presented in Section 3.

For an arbitrary optimization problem, let $F(\cdot)$ denote the set of feasible solutions and $v(\cdot)$ the optimal objective function value. Further, let $\cdot^{LP}$ denote the continuous relaxation of a problem with integer variables.

Lemma 2. $F(P_1^{LP}) = F(P^{LP})$.

Proof. It suffices to show that any solution to P^{LP} satisfies (2b).

From constraints (1b) and Lemma 1 follow that, for every subset $S \subseteq \mathscr{S}$ and every $v_k \in S$, $x(\delta(S)) = 2y(S) - 2x(E(S)) = 1 + y(S) - |S| + y(S) + |S| - 1 - 2x(E(S)) \geqslant 1 + y(S) - |S| + y_{v_k} + 2y(S\backslash\{v_k\}) - 2y(S\backslash\{v_k\}) = 1 + y(S) - |S| + y_{v_k} \geqslant 1 + y(S) - |S|$, so (2b) are satisfied. Here, the first equality is obtained by using Lemma 1. To derive the first inequality, (1b) and the relation $|S| - 1 \geqslant y(S\backslash\{v_k\})$ are used. The last inequality simply relies on the fact that $y_{v_k} \geqslant 0$. Since these relations do not depend on integrality of the variables, the lemma follows. □

The numbers of constraints in (1b) and (2b), respectively, both grow exponentially with $|V|$. The constraints (2b) can be implicitly handled in a column generation scheme, i.e., all solutions to the column generation problem will satisfy this restriction. Below it is shown that constraints (1b) are redundant in problem P_1. However, dropping these constraints may give a weaker continuous relaxation. This issue is discussed in more detail later in this section and in Sections 5.1 and 6.

Lemma 3. *Suppose the vertex degree restrictions* (1a) *and the integrality requirements of the variables are satisfied. Then the constraints* (2b) *imply that* (1b) *are satisfied.*

Proof. We prove that if any of the constraint (1b) is violated, then some constraint (2b) is violated. Suppose $(\bar{x}, \bar{y})$ violates (1b). Using the integrality of the variables, we conclude that there exists a subset $S \subseteq \mathscr{S}$ and a $v_k \in S$ such that $\bar{x}(E(S)) \geqslant 1 + \bar{y}(S) - \bar{y}_{v_k}$. Let $S^0 = \{v \in S : \bar{y}_v = 0\}$ and $S^1 = \{v \in S : \bar{y}_v = 1\}$, so that $\bar{y}(S^0) = 0$ and $\bar{y}(S) = \bar{y}(S^1) = |S^1|$. Using Lemma 1, $\bar{y}(S^0) = 0$ implies $\bar{x}(E(S^0)) = \bar{x}(\delta(S^0)) = 0$, which in turn implies that $\bar{x}(E(S^1)) = \bar{x}(E(S))$ holds. Therefore, the assumption of a violated generalized subtour elimination constraint leads to

$$\bar{x}(E(S^1)) \geqslant 1 + \bar{y}(S^1) - \bar{y}_{v_k}. \tag{3}$$

Thus, $0 \leqslant \bar{x}(\delta(S^1)) = 2\bar{y}(S^1) - 2\bar{x}(E(S^1)) \leqslant 2\bar{y}_{v_k} - 2 \leqslant 0$, i.e. $\bar{x}(\delta(S^1)) = 0$. Here, the first equality follows from Lemma 1 and the second inequality follows from (3).

Consequently, $0 = \bar{x}(\delta(S^1)) \ngeqslant 1 + \bar{y}(S^1) - |S^1| = 1$, and a violated constraint in (2b) has been identified. □

As shown below, the generalized subtour elimination constraints (1b) can be dropped from P_1, without changing the set of integer feasible solutions of P_1 (or P). This gives,

$$\begin{aligned} [P_2] \quad \min \quad & w^{\mathrm{T}}x - r^{\mathrm{T}}y \\ \text{s.t.} \quad & x(\delta(v)) = 2y_v, \quad v \in V\backslash\{v_2\}, \qquad \text{(4a)} \\ & \text{(2a), (2b), (1c)–(1f).} \end{aligned}$$

Theorem 1. $F(P_2) = F(P)$.

Proof. Since the constraints (1b) are dropped from P_1, $F(P_1) \subseteq F(P_2)$ holds. Conversely, $F(P_1) \supseteq F(P_2)$ holds because of Lemma 3. Finally, $F(P_1) = F(P)$ is a consequence of Lemma 2. □

Using similar arguments as in the proof of Theorem 1 the following result is obtained.

Observation 1. $v(P^{LP}) \geqslant v(P_2^{LP})$.

A negative result is that there exist cases where strict inequality is obtained in Observation 1, as is shown in Example 1 below. However, the continuous relaxation of P_2 can be strengthened by adding the subset of the generalized subtour elimination constraints (1b) for which $|S| = 2$:

$$x_e \leqslant y_v, \quad e \in E, \quad v \in \{p(e), q(e)\}, \quad v \neq v_1, \tag{5}$$

where $p(e)$ and $q(e)$ denote the end-vertices of edge e. (Constraints of this form are sometimes called variable upper bounds, or VUB constraints.) Since $y_{v_1} = 1$, the constraints $x_e \leqslant y_{v_1}, e \in \delta(v_1)$, are redundant and therefore not included in (5). The utilization of the constraints (5) is described in Section 3.4. In computations, this set of constraints is convenient because of its limited size. Further, there are reasons to believe that the constraints (5) are strong, because when the knapsack constraint (1c) is removed from P_2, each constraint of type (5) defines a facet of the resulting polytope; this is proved in [3]. (The possibility of using the remaining constraints in (1b) is discussed in Sections 5.1 and 6.)

Let P_3 denote the formulation where the constraints (5) are appended to P_2. Then P_3 can be written as

$$\begin{aligned}
[P_3] \quad \min \quad & w^{\mathrm{T}}x - r^{\mathrm{T}}y \\
\text{s.t.} \quad & x(\delta(v)) = 2y_v, \quad v \in V\backslash\{v_1, v_2\}, && \text{(6a)}\\
& x_e \leqslant y_v, \quad (e, v) \in I, && \text{(6b)}\\
& x(\delta(v_1)) = 2, && \text{(6c)}\\
& \text{(2a), (2b), (1c)– (1f),}
\end{aligned}$$

where

$$I = \{(e, v) | e \in E, v \in \{p(e), q(e)\}, v \neq v_1\}. \tag{7}$$

Note that the degree constraint for v_1 is written separately. This is done in order to highlight the special structure of the set of constraints obtained if the first three set of constraints in P_3 are ignored. We take advantage of this structure in a column generation scheme presented in Section 3.

We close this section by the following observation and a small example.

Observation 2. $\upsilon(P_2^{LP}) \leqslant \upsilon(P_3^{LP}) \leqslant \upsilon(P^{LP}) \leqslant \upsilon(P)$.

Example 1. Consider a complete graph G with eight vertices, vertex revenues $r = (0, 7, 17, 12, 55, 94, 97, 48)$, vertex weights $d = (0, 17, 83, 59, 5, 79, 84, 97)$, total vertex weight limitation $D = 339$, and with Euclidean edge costs

$$\{w_e\} = \begin{pmatrix}
- & 75 & 87 & 84 & 68 & 104 & 93 & 42 \\
 & - & 50 & 27 & 88 & 61 & 61 & 79 \\
 & & - & 24 & 57 & 17 & 11 & 65 \\
 & & & - & 74 & 34 & 35 & 73 \\
 & & & & - & 68 & 54 & 26 \\
 & & & & & - & 14 & 79 \\
 & & & & & & - & 66 \\
 & & & & & & & -
\end{pmatrix}.$$

Figs. 1and 2 show optimal solutions for different formulations. Each circle represents a vertex in the Euclidean plane. On the axes, coordinates are given. Solid edges correspond to $x_e = 1$ and dashed edges to fractional values. An optimal solution to P_2^{LP} is given in Fig. 1(a). In that solution, three constraints of type (5) are violated. Fig. 1(b) shows an optimal solution to P_3^{LP}. In Fig. 2(a), an optimal solution to P^{LP} is shown. Finally, Fig. 2(b) shows an optimal solution to the integer optimization problem P.

To illustrate the relations in Observation 2, this example gives $\upsilon(P_2^{LP}) = -119.65 < \upsilon(P_3^{LP}) = -117.5 < \upsilon(P^{LP}) = -69.16 < \upsilon(P) = -54$. For most instances, the inequalities are strict, as is the case in this example. Further, in this example, the lower bound to $\upsilon(P)$ obtained from $\upsilon(P_3^{LP})$ is not much better than the one from $\upsilon(P_2^{LP})$. Nevertheless, as is pointed out in Section 5, the bound typically becomes significantly better when constraints of type (5) are appended to the formulation.

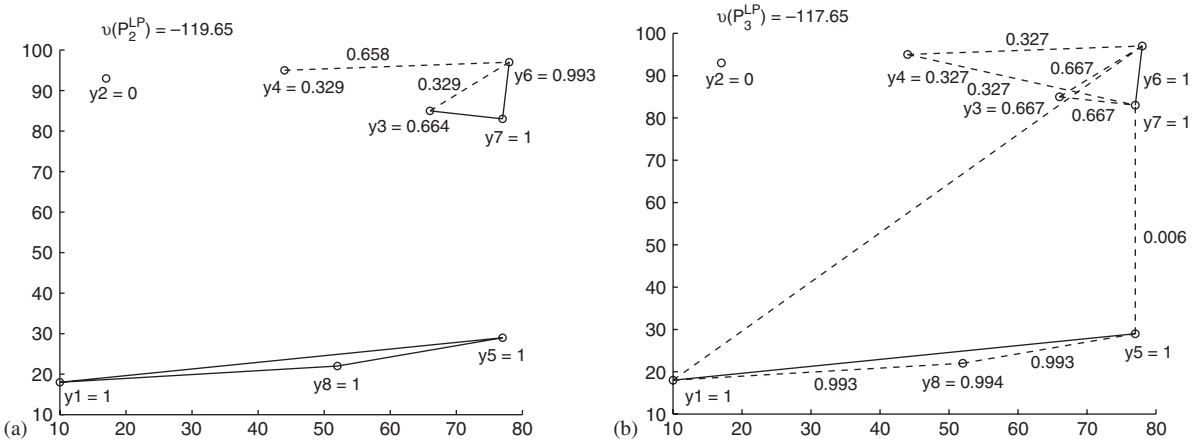

Fig. 1. (a) An optimal solution to P_2^{LP}. (b) An optimal solution to P_3^{LP}.

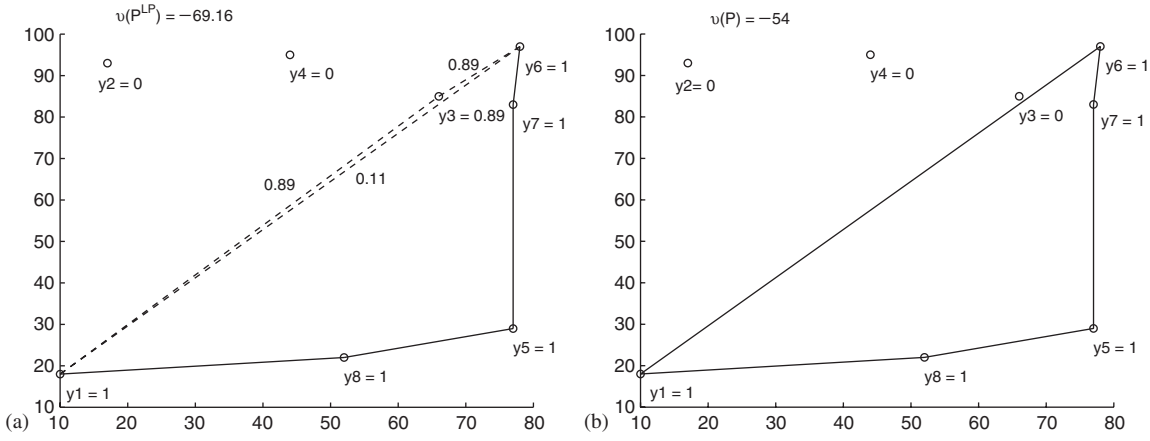

Fig. 2. (a) An optimal solution to P^{LP}. (b) An optimal solution to P.

3. Stabilized column generation

In this section a mathematical decomposition of P_3 is introduced. Based on this decomposition, we propose a column generation scheme which exploits the problem structure. Such schemes are, however, known to suffer from an inherent instability (see e.g. [29, Chapter 15]), in the sense that the values of the dual variables may change significantly between iterations, which in turn may lead to poor convergence behavior.

To improve the stability of the column generation procedure, we incorporate a boxstep restriction [42], which amounts to introducing adaptive bounds on individual dual variables. This technique has recently been used, with good results, in [38], where a column generation procedure for the side constrained traffic equilibrium problem is addressed. A modification similar to the boxstep restriction has been used for stabilizing column generation procedures in [15]. It is applied to three types of applications (airline crew pairing problem, multisource Weber problem, and p-median problems), and it is reported that using stabilized column generation reduces solution time by a factor ranging from 2 to 10, compared to when no stabilization is done.

3.1. Introduction

Consider the formulation P_3 at the end of Section 2. Let $\mathscr{F}$ be the set of points (x, y) that satisfies all but the three first set of constraints of P_3. Suppose the set $\mathscr{F}$ consists of p points ($\mathscr{F}$ is clearly finite), say (x^t, y^t), $t = 1, \ldots, p$. Then the problem P_3 can be restated as a problem P_5, which has the linear programming relaxation

$$[P_5^{LP}] \quad \min \quad \sum_{t=1}^{p} (w^{\mathrm{T}} x^t - r^{\mathrm{T}} y^t)\lambda_t$$

$$\text{s.t.} \quad \sum_{t=1}^{p} \left(\sum_{e \in \delta(v)} x_e^t - 2y_v^t \right) \lambda_t = 0, \quad v \in V \backslash \{v_1, v_2\}, \tag{8a}$$

$$\sum_{t=1}^{p} \left(\sum_{v \in V \backslash \{v_1\}} d_v y_v^t - D \right) \lambda_t \leqslant 0, \tag{8b}$$

$$\sum_{t=1}^{p} (x_e^t - y_v^t)\lambda_t \leqslant 0, \quad (e, v) \in I, \tag{8c}$$

$$\sum_{t=1}^{p} \lambda_t = 1, \tag{8d}$$

$$\lambda_t \geqslant 0, \quad t = 1, \ldots, p. \tag{8e}$$

This problem is recognized as the linear programming relaxation of a Dantzig–Wolfe master problem of P_3. Computational experience shows that many of the constraints (8c) are redundant, and that the problem P_5^{LP} typically becomes heavily degenerate. As discussed at the beginning of Section 3.4, this results in very long solution times. Let P_4^{LP} denote the formulation where all constraints of type (8c) are dropped from P_5^{LP}. It is clear that $v(P_4^{LP}) \geqslant v(P_2^{LP})$ always holds. Furthermore, strict inequality may hold, see the example below. This is due to that the points in $\mathscr{F}$ are not necessarily extreme points in the polytope obtained when the integrality restrictions are removed from the set of constraints that describes $\mathscr{F}$; see also Example 2.

The problem P_4^{LP} has a huge number of columns, even for a relatively small graph. It is therefore practically impossible to find and store all $(x^t, y^t) \in \mathscr{F}$. In Section 3.2, a stabilized column generation procedure for solving P_4^{LP} is introduced. Further, in Section 3.4, a procedure for gradually appending some of the constraints of type (8c) is developed. [Recall that the constraint (8c) directly corresponds to (5).]

3.2. The master problem

Let the dual variables associated with (8a)–(8d) be denoted by α_v, $\beta(\leqslant 0)$, $\mu_{e,v}(\leqslant 0)$ and γ, respectively. The dual boxes used in the stabilized column generation scheme are expressed by the constraints

$$\tilde{\alpha}_v - \varepsilon_\alpha \leqslant \alpha_v \leqslant \tilde{\alpha}_v + \varepsilon_\alpha, \quad v \in V \backslash \{v_1, v_2\}, \tag{9a}$$

$$\tilde{\beta} - \varepsilon_\beta \leqslant \beta \leqslant \tilde{\beta} + \varepsilon_\beta, \tag{9b}$$

where $\tilde{\alpha}$, $\tilde{\beta}$ is the dual solution to the previous master problem (defined below), and ε_α and ε_β are given positive parameters. This means that in each iteration of the column generation process, the dual box is centered around $\tilde{\alpha}$, $\tilde{\beta}$.

This is a modification of the boxstep method [42], in which the box is kept still until the problem over this box is solved to a high prespecified accuracy. In our approach, only one new column is appended before the box is moved. In the approach in [15], a box is also put around the dual iterates, but unlike the original boxstep method, the dual variables are allowed to take values outside the box. This is, however, penalized in the objective function. Our stabilization technique can be viewed as a special case of this approach, putting the penalties to infinity.

Let the dual variables associated with (9) be denoted by $\eta_v^{lb}, \eta_v^{ub}, v \in V\backslash\{v_1, v_2\}$ and by η_{knap}^{lb} and η_{knap}^{ub}, where knap is short for knapsack; these (primal) variables are called artificial. The problem obtained when the artificial columns are appended to P_4^{LP} is denoted *MP* (master problem). Below a restriction of *MP* is introduced where only τ ($< p$) of all non-artificial columns are known.

$$
\begin{aligned}
[RMP_\tau] \quad \min \quad & \sum_{t=1}^{\tau}(w^{\mathrm{T}}x^t - r^{\mathrm{T}}y^t)\lambda_t + \sum_{v\in V\backslash\{v_1\}}((\tilde{\alpha_v} + \varepsilon_\alpha)\eta_v^{ub} - (\tilde{\alpha_v} - \varepsilon_\alpha)\eta_v^{lb}) \\
& + (\tilde{\beta} + \varepsilon_\beta)\eta_{\text{knap}}^{ub} - (\tilde{\beta} - \varepsilon_\beta)\eta_{\text{knap}}^{lb} \\
\text{s.t.} \quad & \sum_{t=1}^{\tau}\left(\sum_{e\in\delta(v)} x_e^t - 2y_v^t\right)\lambda_t + \eta_v^{ub} - \eta_v^{lb} = 0, \quad v \in V\backslash\{v_1, v_2\}, && \text{(10a)} \\
& \sum_{t=1}^{\tau}\left(\sum_{v\in V\backslash\{v_1\}} d_v y_v^t - D\right)\lambda_t + \eta_{\text{knap}}^{ub} - \eta_{\text{knap}}^{lb} \leqslant 0, && \text{(10b)} \\
& \sum_{t=1}^{\tau}\lambda_t = 1, && \text{(10c)} \\
& \lambda_t \geqslant 0, \quad t = 1, \ldots, \tau, && \text{(10d)} \\
& \eta_v^{lb}, \eta_v^{ub} \geqslant 0, \quad v \in V\backslash\{v_1, v_2\}, && \text{(10e)} \\
& \eta_{\text{knap}}^{lb}, \eta_{\text{knap}}^{ub} \geqslant 0. && \text{(10f)}
\end{aligned}
$$

Define

$$\hat{w}_e = w_e - \alpha_{p(e)} - \alpha_{q(e)}, \tag{11a}$$

$$\hat{r}_v = r_v - 2\alpha_v + d_v\beta. \tag{11b}$$

To find a new column with minimal reduced cost, we solve the column generation problem

$$
\begin{aligned}
[CGP] \quad \min \quad & \hat{w}^{\mathrm{T}}x - \hat{r}^{\mathrm{T}}y \\
\text{s.t.} \quad & \text{(6c), (2a), (2b), (1d)– (1f).}
\end{aligned}
$$

Let $(x^{\tau+1}, y^{\tau+1})$ denote an optimal solution to *CGP*. Then the reduced cost for the new column is $\hat{c}_{\tau+1} = \hat{w}^{\mathrm{T}}x^{\tau+1} - \hat{r}^{\mathrm{T}}y^{\tau+1} - \gamma$.

Example 1 (*Continued*). In Fig. 3, two optimal solutions to *CGP* are given for $\alpha_3 = 2.5$; $\alpha_4 = \alpha_5 = \alpha_6 = \alpha_7 = \alpha_8 = \beta = 0$. Note that these solutions only differ in the values of y_3 and y_4. The solutions to *CGP* can be modified into feasible cycles to TSSP_3 (which is the problem defined at the beginning of Section 2.1). This is discussed in Section 4.

Moreover, an optimal solution to P_4^{LP} is obtained when taking a convex combination of these *CGP*-solutions, where both convexity weights equal 0.5. This solution is given in Fig. 4. (Recall that P_4^{LP} is obtained when all constraints (8c) are removed from P_5^{LP}.) Note that in this example, the lower bound from $v(P_4^{LP})$ is strictly better than the bound from $v(P_2^{LP})$, see Fig. 1(a).

In Section 3.3 we describe how *CGP* is solved. The solution to *CGP* is translated into the new column in $RMP_{\tau+1}$, which is then resolved. This solution process is repeated until no further column with negative reduced cost can be generated, which is the case after a finite number of iterations. The master program is always feasible, because of the presence of artificial variables (which are initially the only variables).

Upper and lower bounds to $v(P_4^{LP})$ can be computed during the column generation process. Since *MP* includes more variables (namely the artificial columns) than P_4^{LP}, it is a relaxation of P_4^{LP}, and therefore any lower bound to $v(MP)$ is valid also for $v(P_4^{LP})$. Let $\underline{v}$ denote the best lower bound to $v(MP)$ found so far, and let $\hat{c}_{\tau+1}$ be the reduced cost

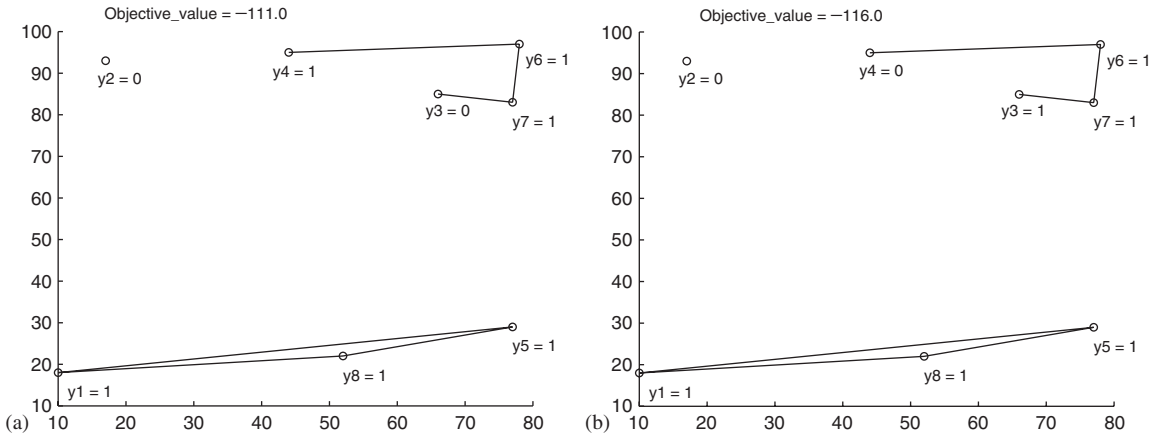

Fig. 3. Two alternative *CGP* solutions for Example 1, obtained for $\alpha_3 = 2.5$; $\alpha_4 = \alpha_5 = \alpha_6 = \alpha_7 = \alpha_8 = \beta = 0$.

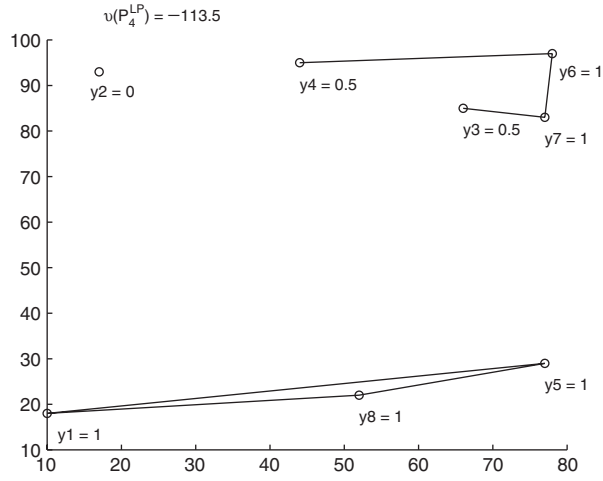

Fig. 4. An optimal solution to P_4^{LP} for the instance in Example 1.

of the newly generated column. Then $\underline{v} := \max\{\underline{v}, \upsilon(RMP_\tau) + \hat{c}_{\tau+1}\}$, see [39]. (Here, $\underline{v}$ is initiated to $-\infty$.) Upper bounds to $\upsilon(P_4^{LP})$ are found more rarely, since such are obtained only when all artificial variables, η, are zero.

3.3. The column generation problem

In this section the column generation problem *CGP*, introduced in Section 3.2, is studied. Below, it is shown that this problem can be efficiently solved, using the concept of a 1-tree with one degree-constraint.

Definition 1. A 1-*tree with one degree constraint*, denoted $C1T$, in an undirected graph $G = (V, E)$, is a set of $|V|$ edges that spans V and such that a designated vertex has a specified degree.

The column generation problem, *CGP*, can be transformed into a problem of finding an optimal solution to a minimum cost $C1T$. Introduce an extended variable vector $\xi = (x'^{\mathrm{T}}, x^{\mathrm{T}})^{\mathrm{T}}$, where x' is the binary vector defined as $x'_v = 1 - y_v$, $v \in V\backslash\{v_1\}$, $x'_{v_1} = y_{v_1} \equiv 1$. Hence, $x'_v = 1$ if vertex v is *not* visited by the cycle, 0 otherwise. This variable

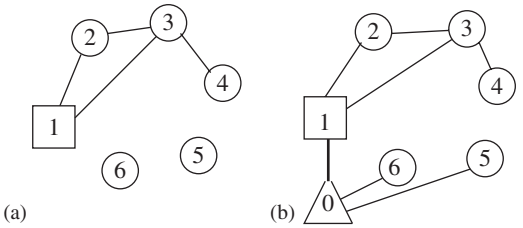

Fig. 5. (a) A feasible solution to *CGP*. (b) The corresponding 1-tree.

substitution has an interpretation in a graph closely related to G. Expand G by introducing a new vertex v_0, called the root. For each vertex $v \in V$, an edge with the endpoints v_0 and v is added, called a root edge and denoted by $e_{v_0,v}$. Then $x'_v = 1$ means that the edge $e_{v_0,v}$ is included in the solution to the minimum cost $C1T$. The cost of the root edge corresponding to v is $w_{e_{v_0,v}} = r_v$, $v \in V$. Denote the expanded graph by $G^0 = (V^0, E^0)$, where $V^0 = V \cup \{v_0\}$ and $E^0 = E \cup \{e_{v_0,v} \mid v \in V\}$. This transformation is also used in [8], with the aim of identifying a strong relaxation of the Steiner tree problem. A similar transformation is used in [19], in a bounding procedure for the prize collecting TSP. See also [55] for a related topic.

Using the transformation above, *CGP* is restated in the graph G^0. Here, $\hat{\psi} = (\hat{r}^{\mathrm{T}}, \hat{w}^{\mathrm{T}})$, where $\hat{w}$ and $\hat{r}$ are given by (11a) and (11b) [or, later, by (16a) and (16b)].

$$
\begin{aligned}
[C1T_{G^0}] \quad \min \quad & \hat{\psi}^{\mathrm{T}}\xi \\
\text{s.t.} \quad & \xi(\delta(v_1)) = 3, && \text{(13a)}\\
& \xi(E^0) = |V^0|, && \text{(13b)}\\
& \xi(\delta(U)) \geqslant 1, \quad U \subset V^0, && \text{(13c)}\\
& x'_{v_1} = 1, && \text{(13d)}\\
& x_e \in \{0, 1\}, \quad e \in E, && \text{(13e)}\\
& x'_v \in \{0, 1\}, \quad v \in V. && \text{(13f)}
\end{aligned}
$$

$C1T_{G^0}$ is the problem of finding a 1-tree in the graph G^0, such that the degree of the depot is fixed to three and $x'_{v_1} = 1$. In Fig. 5(a), a feasible solution to *CGP* is shown, and in Fig. 5(b), the corresponding degree-constrained 1-tree is depicted. The formulation $C1T_{G^0}$ does, in general, *not* have the integrality property, see the counter example in Example 2.

Example 2. Consider the complete graph G^0 in Fig. 6. Those edges that are not shown have a very large cost. Fig. 7(a) shows an optimal solution to $C1T_{G^0}$, and Fig. 7(b) shows an optimal solution to $C1T^{LP}_{G^0}$. Since the objective values are different, the formulation $C1T_{G^0}$ does not have the integrality property. Since $C1T_{G^0}$ is obtained from *CGP* by a simple variable substitution, *CGP* does not have the integrality property either.

The problem $C1T$ constitutes a special case of the minimum weight k-tree with one degree-constraint problem [20]. A k-tree with one degree-constraint in an undirected graph $G = (V, E)$, is defined as a set of $|V| - 1 + k$ edges that spans V and such that a designated vertex has a specified degree. In [20], an $O(|V|^3)$ algorithm for the solution of the minimum weight k-tree problem. The main characteristic of this algorithm is the following. First a minimal weight k-tree problem is solved, ignoring the degree constraint for the depot. Next, edge exchanges are performed in order to attain feasibility with respect to the degree constraint for the designated vertex. In our case, we use the algorithm for the special case when $k = 1$.

3.4. Late inclusion of constraints

In this section we describe how the column generation procedure can be combined with late inclusion of constraints, in order to strengthen the lower bound to TSSP_3 (defined at the beginning of Section 2.1) which is obtained from P_4^{LP}.

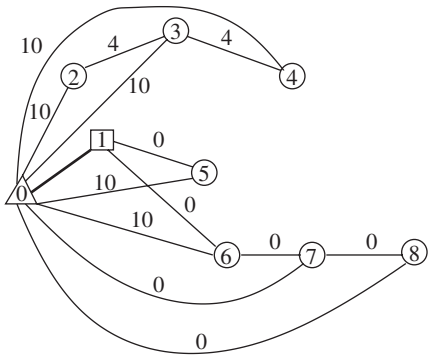

Fig. 6. A graph G^0 with associated edge costs.

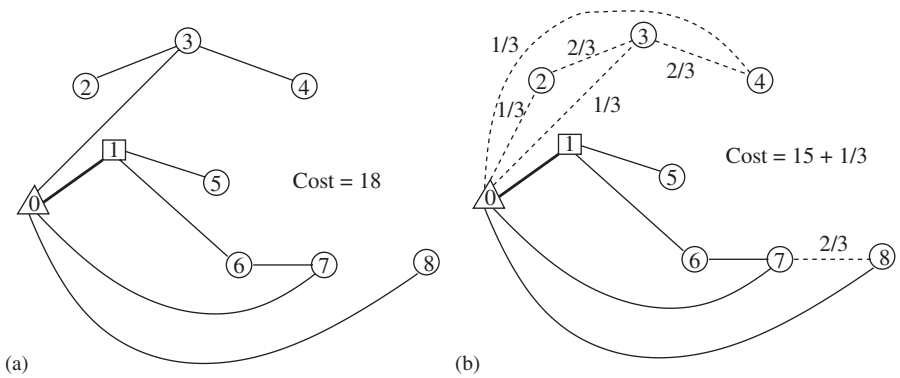

Fig. 7. (a) An optimal integer solution for $C1T_{G^0}$, (b) an optimal solution to the linear programming relaxation for $C1T_{G^0}$.

The first to combine cut and column generation may have been Appelgren [1]. In his work with integer programming methods for vessel scheduling, he identifies a class of valid inequalities, and successively appended constraints from this class to a Dantzig–Wolfe master program. Moreover, these constraints are also explicitly taken into account in the column generation problems. The algorithm is tested on some small test problems. However, he reports that standard branch-and-bound works better.

Combinations of column and constraint generation are also considered in [44,52,46,30], and a profound analysis of combining column and constraint generation is presented in [51]. They point out that a condition for this combination to work well, is that the dual variables associated with the newly generated constraints are not allowed to change the fundamental structure of the objective function in the subproblem(s). Still, even though this condition is satisfied in their application, they report rather disappointing computational experiences. The lower bounds are raised considerably, but the computing times are long, because the reoptimization of the restricted master programs, after that new constraints have been added, requires about the same number of columns generated as for solving the original master program. In Section 5.3.1, we present computational results that resembles the results reported on in [51].

Related works on branch-and-price-and-cut methods are presented in [6,53]. They both report good results, better than those obtained by only performing branch-and-price. A technical review of column generation in integer programming is presented in [56]. In Section 4.4.1 of this paper, the combination of row and column generation is discussed.

Our original master problem, RMP_τ, has $|V|+1$ constraints. The total number of constraints of type (8c) is $(|V|-1)^2$. One strategy might be to add all these constraints to the master problem. This is however likely to cause a large degree of degeneracy. This experience is made in [40], where strong linear programming relaxations for the orienteering problem are studied, and constraints of type (5) are used as valid inequalities. When all these constraints are added to

their linear program, the CPU-time increases very much, because of degeneracy. Instead only constraints of type (5) that are violated at the linear programming solution are added in their work. Our experiences confirms their results. Furthermore, computational tests show (see Section 5.3.1) that less than 2% of the constraints of type (8c) are active close to an optimal solution. Therefore, a scheme with late inclusion of constraints is used, finding constraints of type (8c) as they are needed. Violated constraints are easily identified. First, the current master solution is translated into a (x, y)-solution using $(x, y) = \sum_{t=1}^{\tau} \lambda_t (x^t, y^t)$. Then the constraints of type (8c) are checked by examining each edge. A drawback is that the method needs much memory, since every *CGP*-solution needs to be stored.

Next some notation is introduced to be able to describe how constraints of type (8c) are included into the master problem. Each constraint of type (8c) corresponds to a pair (e, v) in the set I, which is given by (7), and the corresponding dual variable is denoted by $\mu_{e,v} (\leqslant 0)$, see beginning of Section 3.2. In a similar way to (9), dual boxes are expressed by the constraints

$$\tilde{\mu}_{e,v} - \varepsilon_\mu \leqslant \mu_{e,v} \leqslant \tilde{\mu}_{e,v} + \varepsilon_\mu, \quad (e, v) \in I^P, \tag{14}$$

where $\tilde{\mu}$ is a component of the dual solution to the master problem, ε_μ is a parameter and $I^P \subseteq I$ corresponds to the set of constraints currently present in the master problem. The corresponding primal variables are denoted by $\eta^{lb}_{e,v}$ and $\eta^{ub}_{e,v}$. Let MP^I be the master problem where the constraints in I are appended. MP^{I^P} is defined analogously to MP^I and the restriction $RMP^{I^P}_\tau$ is given by

$$\begin{aligned}
[RMP^{I^P}_\tau] \quad \min \quad & \sum_{t=1}^{\tau} (w^{\mathrm{T}} x^t - r^{\mathrm{T}} y^t)\lambda_t + \sum_{v \in V \setminus \{v_1\}} ((\tilde{\alpha_v} + \varepsilon_\alpha)\eta^{ub}_v - (\tilde{\alpha_v} - \varepsilon_\alpha)\eta^{lb}_v) \\
& + (\tilde{\beta} + \varepsilon_\beta)\eta^{ub}_{\mathrm{knap}} - (\tilde{\beta} - \varepsilon_\beta)\eta^{lb}_{\mathrm{knap}} + \sum_{(e,v) \in I^P} ((\tilde{\mu}_{e,v} + \varepsilon_\mu)\eta^{ub}_{e,v} - (\tilde{\mu}_{e,v} - \varepsilon_\mu)\eta^{lb}_{e,v}) \\
\text{s.t.} \quad & \sum_{t=1}^{\tau} (x^t_e - y^t_v)\lambda_t + \eta^{ub}_{e,v} - \eta^{lb}_{e,v} \leqslant 0, \quad (e, v) \in I^P, && \text{(15a)} \\
& \eta^{lb}_{e,v}, \eta^{ub}_{e,v} \geqslant 0, \quad (e, v) \in I^P, && \text{(15b)} \\
& \text{(10a)–(10f).}
\end{aligned}$$

Recall that the dual variables to $RMP^{I^P}_\tau$ are denoted by α_v, $\beta (\leqslant 0)$, $\mu_{e,v} (\leqslant 0)$ and γ, associated with (10a), (10b), (15a) and (10c), respectively. Let $(x^{\tau+1}, y^{\tau+1})$ be an optimal solution to the column generation problem, *CGP*, see Section 3.3. The reduced cost of the new column is $\hat{c}_{\tau+1} = \hat{w}^{\mathrm{T}} x^{\tau+1} - \hat{r}^{\mathrm{T}} y^{\tau+1} - \gamma$, where

$$\hat{w}_e = w_e - \alpha_{p(e)} - \alpha_{q(e)} - \sum_{v:(e,v) \in I^P} \mu_{e,v}, \tag{16a}$$

$$\hat{r}_v = r_v - 2\alpha_v + d_v \beta - \sum_{e:(e,v) \in I^P} \mu_{e,v}. \tag{16b}$$

Initially, no constraints of type (8c) are present in the master program, i.e. $I^P := \emptyset$, and later most violated constraints of type (8c) are appended. If no violated constraint exists, then P^{LP}_5 has been solved to a known accuracy. Since MP^{I^P} is a relaxation of MP^I, each lower bound to $v(MP^{I^P})$ is also a lower bound to $v(MP^I)$, which in turn is a lower bound to $v(P^{LP}_5)$. Upper bounds to relaxations of P^{LP}_5, where the constraints (8c) are missing for $(e, v) \in I \setminus I^P$, are obtained when all artificial variables in $RMP^{I^P}_\tau$ become zero; cf. the description at the end of Section 3.2.

Below the algorithm is given. It terminates when the lower and upper bounds to $v(P^{LP}_5)$ are sufficiently close to each other, or when it is detected that TSSP is solved by the empty cycle or a two-cardinality cycle. (Note below that TSSP_3 is the problem defined at the beginning of Section 2.1.)

procedure *Column_Generation_with_Late_Inclusion_of_Constraints (CGwLIoC)*
begin
input: $G = (V, E)$, $w, r, d, D, \Pi, \iota (0 < \iota < 1)$, $\varepsilon_1, \varepsilon_2, \varepsilon_\alpha, \varepsilon_\beta, \varepsilon_\mu$;
output: A fractional solution (x, y), with objective value at most ε_2 worse than $\upsilon(P_5^{LP})$;
Initialize: $\tau := 1$; $I^P := \emptyset$; $\underline{v} := -\infty$; $\hat{c}_1 := \bar{v}^{I^P} := \upsilon(RMP_0^{I^P}) := gap^I := +\infty$;
Put the center of the dual box at the origin;
repeat
repeat
Solve $RMP_\tau^{I^P}$;
if all artificial variables η are zero **then**
$\bar{v}^{I^P} := \upsilon(RMP_\tau^{I^P})$; (Upper bound obtained.)
Solve *CGP*;
$\underline{v} := \max\{\underline{v}, \upsilon(RMP_\tau^{I^P}) + \hat{c}_{\tau+1}\}$;
if $\underline{v} \geqslant \Pi$ **then** (See Section 2.1.)
Abnormal termination, because of $\upsilon(TSSP_3) \geqslant \Pi$.
$\text{gap}^{I^P} := (\bar{v}^{I^P} - \underline{v})/|\underline{v}|$;
Construct $\text{RMP}_{\tau+1}^{I^P}$, i.e. append new column and re-center dual boxes;
$\tau := \tau + 1$;
until $gap^{I^P} \leqslant \varepsilon_1$;
Compute the set $I^V \subseteq I \backslash I^P$ of violated constraints of type (8c);
if $I^V = \emptyset$ **then**
$\text{gap}^I := \text{gap}^{I^P}$;
$\varepsilon_1 := \iota \varepsilon_1$;
else
Choose a subset $I^C \subseteq I^V$;
Construct $RMP_\tau^{I^P \cup I^C}$;
$I^P := I^P \cup I^C$;
$\bar{v}^{I^P} := +\infty$; $\text{gap}^{I^P} := +\infty$;
until $gap^I \leqslant \varepsilon_2$;
end;

Proposition 1. *The procedure CGwLIoC terminates finitely with either an ε_2-optimal solution to P_5^{LP}, or the conclusion that TSSP is solved by the empty cycle or a two-cardinality cycle.*

Proof (*Sketch*). If $\underline{v} \geqslant \Pi$ occurs, then TSSP is solved by the empty cycle or a two-cardinality cycle, and the procedure terminates.

Since TSSP_3 has a feasible solution and a bounded optimal objective value, this is also true for P_3 and P_5^{LP}. The number of columns and constraints in P_5^{LP} is finite, so that the column generation and the addition of constraints of type (8c) must eventually come to an end. Since no more columns or constraints are needed, this final master problem contains sufficient information to provide solutions to MP^I. Furthermore, the final problem $RMP_\tau^{I^P}$, and its dual, remain unchanged in those iterations, apart from the fact that the dual boxes will change positions.

The procedure can then be interpreted as the search for a maximum of a given piecewise linear and concave function in the dual space, using the boxstep strategy. A basic property of such a function is that its shortest subgradients are bounded away from zero at non-optimal points (that is, that the slope in a steepest ascent direction is not smaller than some value $\rho > 0$, at any non-optimal point). From this property, and the facts that the dual box is of fixed size and is always centered to the previous dual iterate, it follows that the dual objective value will increase with at least a certain minimum amount (which depends of the value of ρ and the box size) in each iteration, unless the box contains an optimal dual solution. Since the optimal dual objective value is bounded from above, one may conclude that the box will contain an optimal dual solution after a finite number of iterations. Let T be the iteration when, for the first time, the

box contains an optimal dual solution to the final master problem. Then the optimal dual solution to $RMP_T^{I^P}$, denoted $u^* = (\alpha^*, \beta^*, \mu^*)$, also solves the dual of the final master problem. In the next iteration, the dual box is re-centered around u^*. Clearly, u^* is then an optimal solution also to the dual of $RMP_{T+1}^{I^P}$, with no active box-constraints. This means that all artificial variables η are zero in the complementary solution to $RMP_{T+1}^{I^P}$. As a consequence, an optimal solution to MP^I with no positive artificial variables has been found. This solution is therefore also an optimal solution to P_5^{LP}. Hence, the procedure terminates finitely with an optimal solution to P_5^{LP}. □

4. Heuristic generation of feasible solutions

We here describe how integer feasible solutions to $TSSP_3$ can be computed from solutions to the relaxed problem, *CGP*. At the end of this section we present the insert and delete heuristic developed by [43]. Contrary to our heuristic the later does not rely on the solution of *CGP*. It is a primal local search method. We shall use also this second heuristic.

In order to construct feasible subtours during the solution process, we have developed a heuristic that manipulates solutions to *CGP* in order to achieve feasibility with respect to the relaxed constraints. The heuristic is designed with the aim of introducing as small changes as possible into the *CGP* solution. It consists of five main parts. In the first part vertices are removed in order to achieve feasibility with respect to the knapsack constraint. Removing vertex v corresponds to putting $y_v := 0$. In the second part, edges connected to vertices v such that $y_v = 0$ are deleted. In the third part, edges are removed to make each vertex degree less or equal to 2. These three steps result in one or many disconnected paths. In the fourth part, these disconnected paths are connected into a cycle. Finally, in the fifth part, we can include a TSP 3-OPT [41] tour improvement procedure if we want to. This is a local search routine that tries to improve the sequence in which the vertices are visited. Since this is relatively time-consuming, we make it optional.

Below the heuristic procedure is described in detail. It has proved to quickly give good feasible solutions. Consider the *CGP* solution (x^{CGP}, y^{CGP}). Let $S^{CGP} = \{v \in V \mid y_v^{CGP} = 1\}$ and $E^{CGP} = \{e \in E \mid x_e^{CGP} = 1\}$. The edge set $\delta^{CGP}(v)$ is the set of edges in E^{CGP} having v as one end-vertex. A vertex set S is said to be feasible if $d(S) \leqslant D$, where $d(S) = \sum_{v \in S} d_v$.

---

procedure *Manipulate_CGP_Solution (MCGPS)*
begin
input: graph $G = (V, E)$, d, $\bar{r}$, $\bar{w}$, S^{CGP}, E^{CGP}; use_3-OPT (true/false)
output: A feasible vertex set S^h, edge set E^h representing a cycle over vertices in S^h and c^h, the total net cost of the cycle;
while $d(S^{CGP}) > D$ **do** Remove from S^{CGP} vertex $v^{\min} = \arg\min_{v \in V \setminus v_1} \{\bar{r}_v\}$;
$S^h := S^{CGP}$;
for all vertices $v \in V \setminus S^h$ **do** Remove from E^{CGP} all edges in $\delta^{CGP}(v)$;
for all vertices $v \in V \setminus \{v_1\}$ **do**
  while $|\delta^{CGP}(v)| > 2$ **do**
    Remove from E^{CGP} edge $e^{\max} = \text{argmax}_{e \in \delta^{CGP}(v)} \{\bar{w}_e\}$;
Let $G^k, k = 1, \ldots, K$ denote the disconnected components (paths) given by S^{CGP} and E^{CGP};
if $K = 1$ **and** the component defines a cycle **then** $E^h := E^{CGP}$;
else
  if one component defines a subcycle **then** remove from E^{CGP} one edge of that subcycle;
  Connect the components into a cycle, by first performing a graph search and then adding K edges $e \in E \setminus E^{CGP}$;
if use_3-OPT **then** Apply the TSP 3-OPT routine to the cycle;
Let E^h be the set of edges in the improved cycle and c^h the total net cost;
end;

The graph search which is performed in the procedure *MCGPS*, heuristically solves the problem of matching the end-vertices of the paths together, i.e. those vertices with degree less than or equal to one. It is a greedy heuristic, since each time we choose a connection from a current component, the cheapest one with respect to the edge costs $\bar{w}$ is chosen.

There is a number of other heuristic approaches to $TSSP_3$. One algorithm that seems to give good result in practice is the insert/delete algorithm developed by [43]. Their algorithm takes any cycle as input. This cycle may violate the knapsack constraint (1c). The general idea is to improve the current cycle by adding to the cycle or deleting from the cycle exactly one vertex in each iteration. If the current cycle is infeasible with respect to the knapsack constraint, an improved cycle is defined as a cycle less infeasible. On the contrary, if the current cycle is feasible, an improved cycle is defined as a feasible cycle with smaller cost. The algorithm continues until no further improvement can be achieved. To apply this heuristic, we first make a simple transformation of our knapsack constraint, which is expressed on vertices, into a knapsack constraint expressed on the edges.

5. Experimental results

In this section we present computational results for a set of test problems. The numerical experiments are divided into three parts. In Section 5.1 some lower bounds are numerically computed, based on different mathematical formulations. In Section 5.2 we study the effects of varying some parameters for the procedure *CGwLIoC* on a single problem instance. The aim of this section is to illustrate the behavior of the algorithm for varying choices of parameter values. In Section 5.3 computations are performed on a large set of instances, evaluating the impact of the constraint (8c) [that corresponds directly to (5)] in terms of obtaining better lower bound to $\upsilon(P)$ and the performance of the procedure *CGwLIoC*. Results are also presented for the heuristic *MCGPS*.

In the procedure *CGwLIoC*, the restricted master $RMP_\tau^{I^P}$ is solved by the dual simplex method using CPLEX 6.5 Simplex callable library, and the problem *CGP* is solved by a Fortran-code in [20]. The remaining part of the procedure is implemented in C, which is also the case for the heuristic procedure *MCGPS* described in Section 4. The Insert/Delete heuristic is a Fortran-code developed by J. Mittenthal and C.E. Noon, see the end of Section 4. All computations are performed on a Sun Ultrasparc 2/2200.

The edge cost matrix is based on Euclidean distances between the vertices. Two subgroups of problems are generated in which the values of the vertex revenues and the values of the vertex weights are correlated and not correlated, respectively. This is in order to observe the difference (if any) in the results for the test problems; we expected that there would be a significant difference.

All test problems are randomly generated according to the following distributions. Each vertex $v \in V$ has two coordinates; the coordinates are uniform random integers in range (1,100). The edge costs w_e are Euclidean, and the vertex revenues r_v are uniform random integer in range (1,$r_{\max}$). The vertex weights, d_v, are uniform random integers in range (1,100) if uncorrelated to r_v; if correlated, d_v is computed as $\gamma_v r_v$, where γ_v are drawn from a uniform distribution with a range of (0.7, 1.3). In all test problems the right hand side of the knapsack constraint (1c) is chosen as $D = \lfloor \alpha \sum_{V\setminus\{v_1\}} d_v \rfloor$, where $\alpha = 0.2, 0.5, 0.8$.

5.1. Comparison of lower bounds to the optimal value

In this subsection we consider various linear programming relaxations of $TSSP_3$ (defined at the beginning of Section 2.1), and make numerical comparisons of lower bounds to $\upsilon(P)$. We focus on how constraints (2b) and (5) affect the lower bound.

First, new notation is introduced. Let P_0^{LP} denote the problem that is obtained if the generalized subtour elimination constraints (1b) are dropped from P^{LP}. Further, let $P_{(2b)}^{LP}$ denote the problem where the constraints (2b) are appended to P_0^{LP}, and define $P_{(5)}^{LP}$ and $P_{(2b,5)}^{LP}$ analogously. Note that $P_{(2b)}^{LP}$ is equivalent to the problem P_2^{LP}, since the constraint $x(E) = y(V)$ is redundant in P_2^{LP}. For the same reason, $P_{(2b,5)}^{LP}$ is equivalent to P_3^{LP}. Further, define the following ratios:

$$k_1 = \frac{\upsilon(P_{(2b)}^{LP}) - \upsilon(P_0^{LP})}{\upsilon(P^{LP}) - \upsilon(P_0^{LP})},$$

$$k_2 = \frac{\upsilon(P_{(5)}^{LP}) - \upsilon(P_0^{LP})}{\upsilon(P^{LP}) - \upsilon(P_0^{LP})},$$

$$k_3 = \frac{\upsilon(P_{(2b,5)}^{LP}) - \upsilon(P_0^{LP})}{\upsilon(P^{LP}) - \upsilon(P_0^{LP})}.$$

Table 1
Average results for test problems with $|V| = 16$ and $r_{\max} = 100$, and vertex weights that are not correlated to vertex revenues

α	k_1 (%)	k_2 (%)	k_3 (%)
0.2	2.9	35.1	48.3
0.5	3.9	35.4	45.3
0.8	10.5	34.0	45.0

Table 2
Results for the five instances, with $\alpha = 0.8$

Instance	k_1 (%)	k_2 (%)	k_3 (%)
1	9.9	64.2	71.6
2	1.7	19.7	25.1
3	36.7	4.1	45.2
4	0.0	19.6	19.6
5	4.1	62.2	63.6

Each of these ratios measures how large portion of the gap between $\upsilon(P^{LP})$ and $\upsilon(P_0^{LP})$ that is closed, when a set of constraints is appended to the problem P_0^{LP}.

In Table 1, numerical results are presented as an average of five instances for each value of α. In these test problems, $|V| = 16$ and $r_{\max} = 100$, and the vertex weights are not correlated to the vertex revenues. In the problems $P_{(2b)}^{LP}$, $P_{(2b,5)}^{LP}$ and P^{LP}, the number of constraints grows exponentially with $|V|$. Therefore, small test problems are chosen in this experiment.

In Table 2 detailed results are presented for the five instances with $\alpha = 0.8$.

It is of course impossible to draw any general conclusions based on these few numerical results. Some details in Tables 1 and 2 can however be highlighted. We first observe that for these test problems, almost 50% of the gap between $\upsilon(P^{LP})$ and $\upsilon(P_0^{LP})$ is closed when the constraints (2b) and (5) are appended to P_0^{LP}. Second, in most cases k_2 is larger than k_1, i.e. $\upsilon(P_{(2b)}^{LP}) > \upsilon(P_{(2b)}^{LP})$ holds. It should be noted that this is not always the case, as can be seen from the third instance in Table 2. Third, the problem $P_{(2b,5)}^{LP}$ sometimes provides a significantly stronger lower bound than $P_{(2b)}^{LP}$ or $P_{(5)}^{LP}$. We conclude that the constraints (2b) or (5) might separately strengthen the lower bound significantly. Further, the combination of these groups of constraints might be significantly stronger than each of them separately. To obtain a strong lower bound both constraints (2b) and (5) should therefore be taken into account. These conclusions provide some justification for our approach, where the constraints (2b) are included in the column generation problem *CGP*, whereas the constraints (5) are appended to the master program as they are violated.

5.2. *Impact of algorithm parameters*

In order to calibrate the algorithm parameters of the procedure *CGwLIoC* we have performed a large number of test runs with different settings on a large number of test problems. In this subsection we illustrate the general observations made on the performance of the solution procedure. For this illustration we use a specific test problem that is representative for the average behavior of the solution procedure; this problem has $|V| = 50$, $r_{\max} = 100$, $\alpha = 0.2$, and vertex weights that are not correlated to the vertex revenues. In the first three experiments of Section 5.2, no violated constraints of type (8c) are appended, i.e. only the inner repeat–until loop of the procedure *CGwLIoC* is executed. In Section 5.2.4, experiments with appending constraints of type (8c) are evaluated.

In the stabilized column generation scheme, the sizes of the dual boxes are important; this fact is established in the second experiment. The dual boxes are expressed by (9) and (14), given ε_α, ε_β and ε_μ. An algorithm parameter denoted by box_size is introduced, and $\varepsilon_\alpha := \varepsilon_\beta := \varepsilon_\mu := box\_size$. In Section 5.2.2, a suggestion for an adaptive rule to change the size of the box during the computations is discussed. The box is initially centered at the origin.

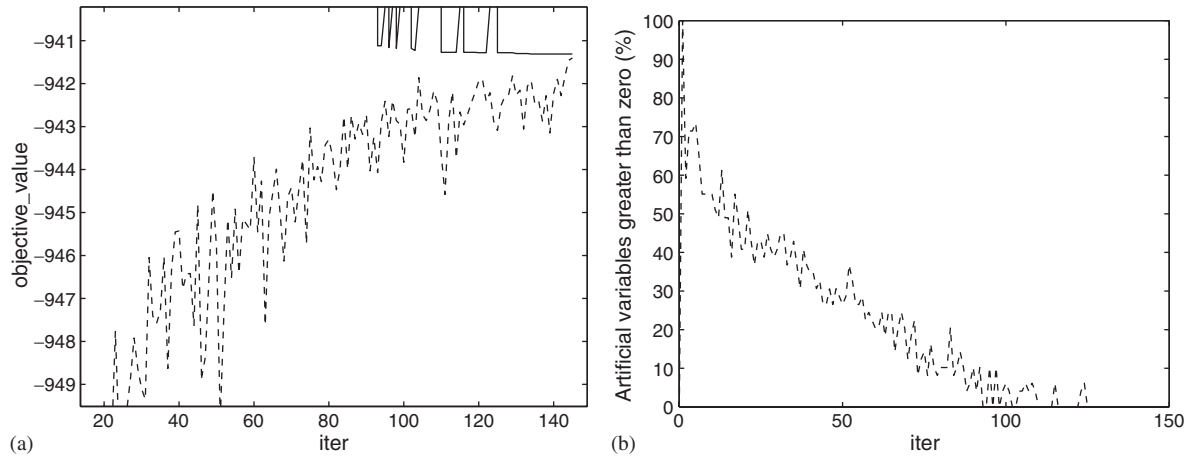

Fig. 8. Behavior of the column generation procedure for $box\_size = 0.5$ and $\varepsilon_1 = 0.01\%$. In (a), candidate values for lower and upper bounds are shown. Note that in the first 94 iterations, no upper bound is known. In (b), the portion of positive artificial variables is plotted.

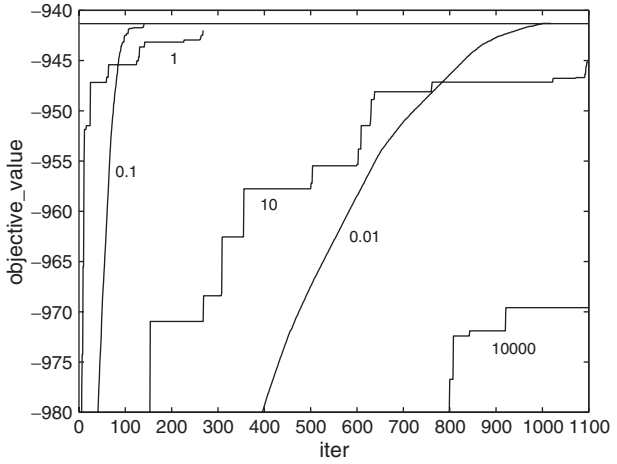

Fig. 9. Lower bound for alternative box sizes. Here, $\varepsilon_1 = 0.1\%$.

5.2.1. Behavior of the column generation procedure

The behavior of the column generation procedure for the choice of $box\_size = 0.5$ and $\varepsilon_1 = 0.01\%$ is shown in Fig. 8. The lower line in Fig. 8(a) corresponds to $\upsilon(RMP_\tau) + \hat{c}_{\tau+1}$, i.e. the candidate values for a lower bound $\underline{\upsilon}$ to $\upsilon(MP)$, see the end of Section 3.2. The upper line shows an upper bound to $\upsilon(P_4^{LP})$.

For each dual variable, there are two dual box-constraints and therefore two primal artificial variables. Because of complementarity, at most one of these two artificial variables can be positive in an optimal basis for a master program. The ratio (#positive artificial variables)/48 is in Fig. 8(b) plotted for each iteration. Here the denominator is the number of dual variables α_v, β. It can be seen that the number of positive artificial variables decreases rather nicely as the number of iterations increase.

In iteration number 95 all artificial variables reach the value zero. This means that an upper bound to $\upsilon(P_4^{LP})$ is obtained. Since the gap is greater than ε_1, more columns are generated until the gap becomes smaller than ε_1.

5.2.2. Various box sizes

In this subsection, results for the procedure *CGwLIoC* for different values of the parameter box_size are compared. In Fig. 9 the upper horizontal line corresponds to $\upsilon(P_4^{LP}) = -941.33$. Also, the lower bound $\underline{\upsilon}$ is plotted for different values of box_size. The computations are terminated when the gap is smaller than $\varepsilon_1 = 0.1\%$.

Table 3
Results for different values of box_size, $\varepsilon_1 = 0.1\%$

box_size	$\#iter$	$\#simplex$	$av\_simplex$	t (s)
0.01	1018	3127	3.1	29.8
0.1	141	2409	17.1	2.8
1	268	6889	25.7	6.8
10	1705	52416	30.7	106.2
10, 000	2599	62447	24.0	225.9

As can be seen from Fig. 9, the choice of the parameter box_size is crucial. For small values, small dual steps are taken. If box_size is very small (0.01) the lower bound slowly crawl upwards. For large values of the parameter box_size large dual steps are taken. In this case, the procedure has an unstable behavior. When box_size is equal to 10,000 we obtain a situation close to the case where no dual box is used.

It can be concluded that the modified boxstep method is very important for the practical convergence rate of the column generation procedure. Furthermore, it is important to choose a suitable size of the box, as indicated also by the CPU-times in Table 3. This table shows some additional data about the runs in Fig. 9. The value of $av\_simplex$ equals the average number of simplex iterations for each master program and t is the total CPU-time. The importance of choosing a suitable value of box_size is further discussed at the end of this section.

Compare the cases $box\_size = 0.01$ to $box\_size = 10$ in Table 3. In the first case, the dual box is small, and only a small number of simplex iterations is needed to reoptimize the dual. In the second case, the box is large, and more work is needed for reoptimization when a new column has been appended to the master problem. The total number of simplex iterations performed is about 16 times larger than for $box\_size = 0.01$.

For $box\_size = 10,000$, the box becomes redundant and large dual steps are taken. This gives an unstable behavior, as can be seen in Fig. 9, and often many columns are generated before the lower bound $\underline{v}$ is improved, so that large master programs must be solved. The problem of working with large master programs can to some extent be dealt with by dropping columns from the master program. This issue is discussed in the next subsection.

A complicating fact is that a good choice of the value of box_size is dependent on characteristics of the problem instance. Instead of calibrating box_size for all groups of test problems, an adaptive rule for changing box_size during the algorithm is developed. This rule depends on the number of times that the lower bound $\underline{v}$ is updated. For example, study $\underline{v}$ in ten consecutive iterations. If it has been increased eight or more times, we suspect that the box is too small and increase the box by $increase\_factor$. If, on the other hand, $\underline{v}$ has been improved only zero or one time, we guess that the value of box_size is too large and decrease it by $decrease\_factor$.

Finally in this subsection, the progress of the algorithm in the dual space in terms of distance to the dual solution is considered. Let $\alpha^\star$ denote optimal values for the dual variables corresponding to the vertex degree restrictions. In Fig. 10 the Euclidean distance from the current dual solution to $\alpha^\star$ is plotted for each iteration. When the box is small, the distance decreases monotonically with the number of iterations, see Fig. 10(a). When the box is large, the behavior is unstable, as shown in Fig. 10(b). Observe that the two figures have different scales, both on the horizontal and vertical axes. It should also be noted that $\alpha^\star$ is not unique, due to primal degeneracy. For the runs in Fig. 10 the final α solution is used as $\alpha^\star$. In Fig. 10(a), $\|\alpha^\star\| = 62.6$ and in Fig. 10(b), $\|\alpha^\star\| = 82.5$.

Observations similar to those in Fig. 10 are made in [31]. They study Lagrangian duality applied to vehicle routing with time windows. Their Lagrangian dual problem is solved in two phases: In the first phase, a cutting plane algorithm with a trust-region stabilizing device (box) is used. In the second phase, the trust-region is removed, which turns the method into a Dantzig–Wolfe algorithm.

5.2.3. Dropping columns

In this subsection the impact of sometimes dropping columns from the master programs is discussed. With an exception for the very first master program, each new master is solved by a reoptimization of the previous master. Still, if there are too many columns in the master, the reoptimization can be computationally burdensome. Therefore, the impact of dropping columns from the master programs is studied. Two alternative strategies are implemented and evaluated. In both cases, let $drop\_int$ denote the dropping interval. Every $drop\_int$ iteration columns are removed from the restricted master.

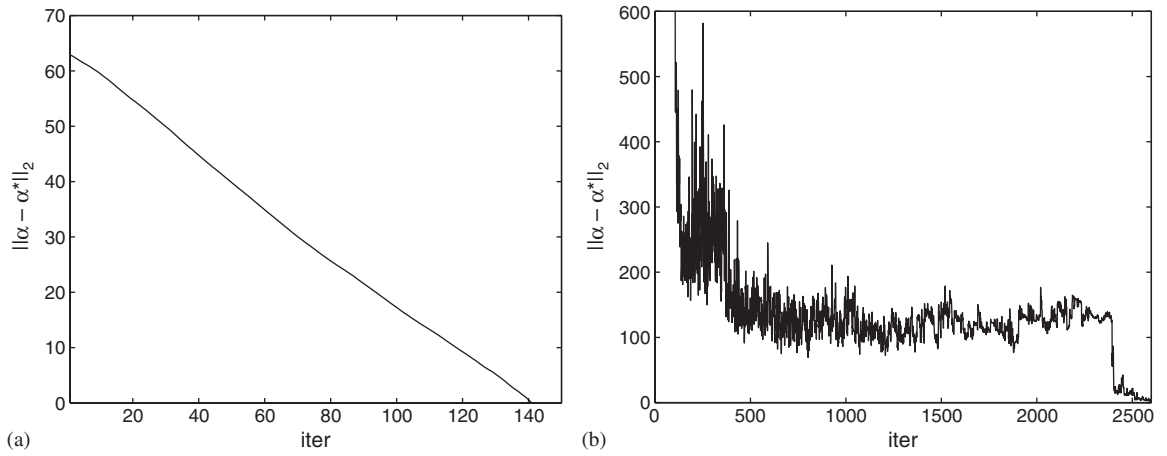

Fig. 10. Euclidean distance to optimal dual solution for two of the runs shown in Fig. 9. In (a), $box\_size = 0.1$ and in (b), $box\_size = 10,000$.

Table 4
Effect of using different values of $drop\_lev$. $drop\_int = 200$, $\varepsilon_1 = 1.0\%$ and $box\_size = 5$

$drop\_lev$ (%)	$\#drop\_col$	$gap^{\emptyset}$ (%)	$iter$	$\#simplex$	t (s)
0	0	0.99	554	80077	78.2
20	65	0.98	456	66059	52.4
40	125	0.97	485	65478	49.9
80	419	0.93	648	80390	54.7
100	1267	0.98	1441	150104	111.2

Table 5
Effect of using different values of $max\_iter\_not\_basic$. $drop\_int = 200$, $\varepsilon_1 = 1.0\%$ and $box\_size = 5$

$max\_iter\_nonbasic$	$\#drop\_col$	$gap^{\emptyset}$ (%)	$iter$	$\#simplex$	t (s)
1000	0	0.99	554	80077	78.2
200	252	0.93	558	77848	66.9
100	346	0.94	618	84269	66.6
50	189	0.95	453	61987	42.4
30	207	1.00	437	56187	42.7
20	429	0.99	605	74406	48.4
10	661	0.99	857	101968	68.8

In the first implementation, the number of columns dropped depends of the total number of columns in the master and the value of $drop\_lev$, the drop level. This number indicates the portion of columns with positive reduced cost that are removed, chosen among the non-artificial columns with largest reduced cost. In the second implementation, columns are dropped based on another criterion. For every non-artificial variable, a number is stored, telling when (in which iteration) it was most recent a basic variable. If the difference between the number of the current iteration and that number is larger than a certain threshold, denoted by $max\_iter\_nonbasic$, the corresponding variable (column) is dropped.

In Tables 4 and 5, $\#drop\_col$ indicates the total number of columns dropped during the procedure. These tables illustrate the performance of the procedure *CGwLIoC* for different values of $drop\_lev$ and $max\_iter\_nonbasic$, respectively, choosing $drop\_int = 200$, $\varepsilon_1 = 1.0\%$ and $box\_size = 5$. Other values of $drop\_int = 200$, for instance 50, 100 or 400 give similar results.

Table 6
Effect of adding constraints in (8c), varying $activate\_lev$. $\varepsilon_1 = \varepsilon_2 = 0.1\%$, and $activate\_limit = 30$. $box\_size = 0.2$. No columns are dropped

$activate\_lev$ (%)	gap^I (%)	$iter$	B (%)	$\#simplex$	t (s)
20.00	0.01	867	0.53	52896	102.8
40.00	0.01	696	0.57	51378	79.5
80.00	0.05	380	0.57	28200	29.5
100.00	0.01	358	0.65	27311	27.2

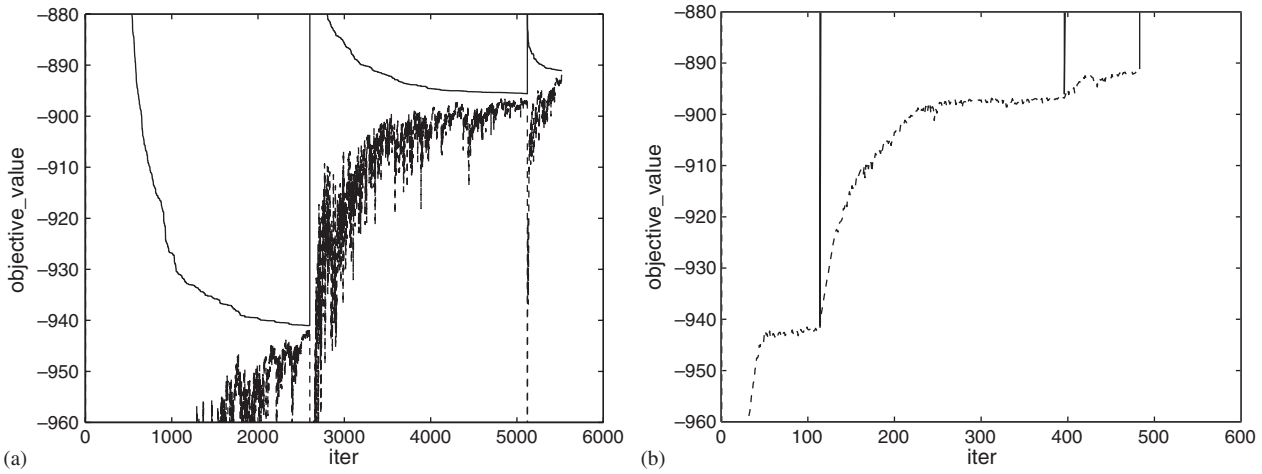

Fig. 11. $\varepsilon_1 = \varepsilon_2 = 0.1\%$, $activate\_lev = 100\%$, $activate\_limit = 30$. No columns are dropped. In (a), $box\_size = 10,000$ and in (b), $box\_size = 0.2$.

The results in Tables 4 and 5 indicate that computational time can be saved if columns are dropped. Nevertheless, if we are too aggressive, i.e., if too many columns are removed too often, the computational time will actually increase, compared to when no columns at all are dropped. This is the case in the last row of Table 4.

5.2.4. Adding constraints of type (8c)

In this subsection some algorithm parameters dealing with the late inclusion of constraints of Section 3.4 are introduced. In Table 6, $activate\_lev$ denotes the portion of the most violated constraints of type (8c) that are appended to the master. Also, $activate\_limit$ is a parameter that gives an upper limit on the number of appended constraints in one iteration. The number

$$B = |I^P|/|I| \tag{18}$$

is the portion of constraints of type (8c) that have been appended to the master at termination.

From Table 6, it seems best to append 100% of the violated constraints in the set I^V to the restricted master program. In our experiments, this has been a good strategy also for other test problems, when using a small value on ε_1. If ε_1 is large the master solution will not be so accurate and the risk of appending constraints that are not so useful increases.

Moreover, it can be observed in Table 6 that gap^I is much smaller than ε_2. This depends on the fact that the size of the box is quite small. When the dual box becomes inactive (see last paragraph of Section 3.2), good lower and upper bounds are obtained and the inner repeat–until loop of *CGwLIoC* is leaved. An example of this behavior is illustrated in Fig. 11(b). Finally, it can be noticed that the total number of constraints of type (8c) that has been appended to the master is small, compared to $|I|$, i.e. the total number of constraints of type (8c). For the test problem in Table 6, less than 0.7% are activated. Still, the last master program does not violate any of the constraints of type (8c).

Fig. 11(a) and (b), display results when the procedure *CGwLIoC* is executed for $box\_size = 10,000$ and 0.2, respectively. Note that the horizontal axis have different scale. After adding 2599 columns in Fig. 11(a), $gap^{\emptyset}$ becomes smaller than ε_1, c.f. last row of Table 3. From iteration 1 to 2599, the lower curve displays candidate values for lower

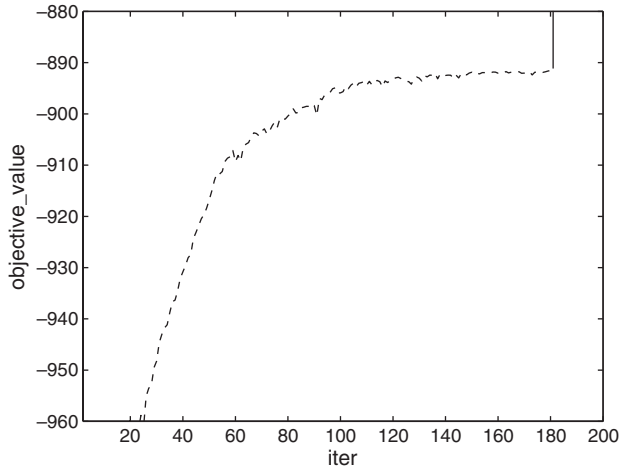

Fig. 12. $\varepsilon_1 = \varepsilon_2 = 0.1\%$, $activate\_lev = 100\%$, $activate\_limit = 30$ and $box\_size = 0.2$. No columns are dropped. 16 constraints of type (8c) are appended to the first master program.

bound $\underline{v}^{\emptyset}$ to $v(MP)$ and the upper curve shows upper bound. When constraints of type (8c) are appended, a more restricted optimization problem is obtained, with a higher optimal objective value.

When $box\_size = 10,000$ the execution time is about 1200 seconds and for $box\_size = 0.2$ the execution time is 29.1 s. In the latter case, the first repeat–until loop is executed in 2.5 s. This example shows that stabilization can reduce solution time with a factor of 40.

Next, consider again Fig. 11(b). During the execution of the first repeat–until loop, 114 columns are generated. Then some of the constraints of type (8c) are appended to the master. Surprisingly, almost 300 new columns are generated, which is rather counter intuitive. Our guess was that adding just a subset of the constraints of type (8c) would not make the problem that much harder.

One might think that this phenomenon is explained by difficulties with the choice of the value of box_size. This, however, is not the case. Many different choices of box_size have been tried, without solving the second phase significantly faster.

In Fig. 11(b), totally 16 constraints of type (8c) are appended to the master. Another experiment was performed, where these 16 constraints were appended to the very first master. The result is shown in Fig. 12. As can be seen, no more constraints of type (8c) was appended this time. Here 181 iterations are executed and the CPU-time is 10.9 s. This can be compared to the time for executing the first repeat–until loop in Fig. 11(b), which is 2.5 s. The conclusion must be that the constraints of type (8c) are hard to handle in a column generation scheme. At the end of Section 5.3.1, this issue is further discussed.

5.3. Computational experience with the overall method

In the first part of this section, we investigate the performance of the procedure *CGwLIoC*. Also, some properties of different groups of test problems are discussed, dealing with gap between upper and lower bounds. Specially, we study how the adding of the constraints of type (8c) makes the gap smaller. In the second part, the heuristic algorithm *MCGPS* of Section 4 for finding integer feasible solution is tested. Specially, we investigate if introducing the constraints of type (8c) into the column generation scheme makes the quality of the heuristic cycles better.

5.3.1. Numerical experiments using CGwLIoC

We first recall notation that was introduced earlier in this section. The value $\underline{v}^{\emptyset}$ denotes the best lower bound obtained for $v(MP)$. Correspondingly, $\underline{v}^{I}$ denotes the best lower bound obtained for $v(MP^I)$. Furthermore, let $\bar{z}_P$ denote the objective value of the best known integer feasible solution to P; this value is computed as follows. An initial candidate value for $\bar{z}_P$ is obtained from the Insert/Delete heuristic. Then, at each iteration of the procedure *CGwLIoC*, the heuristic

Table 7
Results from the procedure *CGwLIoC* for uncorrelated problems

α	$\|V\|$	$r_{\max} = 40$						$r_{\max} = 100$					
		q_1 (%)	q_2 (%)	q_3 (%)	t_i (s)	t (s)	B (%)	q_1 (%)	q_2 (%)	q_3 (%)	t_i (s)	t (s)	B (%)
0.2	50	65.5	45.9	31.0	5	45	0.9	7.8	4.4	43.7	4	34	0.6
	100	23.7	15.9	36.3	34	427	0.3	3.1	1.9	36.9	34	219	0.2
0.5	50	24.6	15.2	38.7	9	125	1.1	5.0	3.4	33.5	7	36	0.5
	100	9.5	6.2	37.1	74	944	0.4	1.9	1.4	29.6	60	228	0.1
0.8	50	18.0	11.5	36.0	14	125	1.0	2.6	1.6	39.7	10	40	0.4
	100	6.1	3.7	39.2	100	721	0.2	1.3	0.9	33.8	94	428	0.1

Table 8
Results from the procedure *CGwLIoC* for correlated problems

α	$\|V\|$	$r_{\max} = 40$						$r_{\max} = 100$					
		q_1 (%)	q_2 (%)	q_3 (%)	t_i (s)	t (s)	B (%)	q_1 (%)	q_2 (%)	q_3 (%)	t_i (s)	t (%)	B (%)
0.2	50	183.5	140.4	23.2	1	28	0.9	36.6	28.2	23.1	2	33	0.9
	100	69.8	55.6	20.6	26	371	0.4	16.7	12.5	24.7	28	466	0.4
0.5	50	51.9	36.5	30.7	6	94	1.4	13.1	9.1	31.0	5	91	1.3
	100	25.1	18.0	28.6	66	1919	0.6	7.2	4.7	35.3	63	1401	0.5
0.8	50	24.2	15.7	36.2	10	150	1.2	6.6	4.2	36.6	9	134	1.1
	100	12.4	8.2	34.2	95	1651	0.5	4.3	2.9	32.7	89	1218	0.4

MCGPS is executed, giving a new candidate value. The lowest of all these candidate values defines the final value of $\bar{z}_P$.

Consider the ratios q_1, q_2 and q_3 defined below. Here q_1 measures the relative gap between $\underline{v}^{\emptyset}$ and $\bar{z}_P$ and q_2 the relative gap between $\underline{v}^I$ and $\bar{z}_P$. Finally, q_3 is the portion of the difference between $\bar{z}_P$ and $\underline{v}^{\emptyset}$ that is closed through the addition of the constraints (8c) to the master program.

$$q_1 = \frac{\bar{z}_P - \underline{v}^{\emptyset}}{|\bar{z}_P|}, \tag{19a}$$

$$q_2 = \frac{\bar{z}_P - \underline{v}^{I}}{|\bar{z}_P|}, \tag{19b}$$

$$q_3 = 1 - \frac{q_2}{q_1} = \frac{\underline{v}^{I} - \underline{v}^{\emptyset}}{\bar{z}_P - \underline{v}^{\emptyset}}. \tag{19c}$$

For a given group of test problems, we know a priori that $q_1 \geqslant q_2$. This follows immediately from (19a) and (19b) and the fact that $\underline{v}^I \geqslant \underline{v}^{\emptyset}$.

For the runs in this subsection, $\varepsilon_1 = \varepsilon_2 = 0.1\%$. The value of box_size is updated during the algorithm using the adaptive rule described in Section 5.2.2. After some initial calibration, the parameters $increase\_factor = \frac{11}{10}$ and $decrease\_factor = \frac{10}{11}$ were chosen. Initially, box_size equals 0.2. Moreover, columns are dropped according to the first strategy described in Section 5.2.3, using $drop\_int = 100$ and $drop\_lev = 50\%$. When violated constraints of type (8c) are appended to the master program, $activate\_lev = 100\%$ and $activate\_limit = 30$. Results are presented for problem where $|V| = 50$ and $|V| = 100$.

For both uncorrelated and correlated problems (Tables 7 and 8) we observe that the value of q_1 strictly decreases when any of the instance parameters $|V|$, α or $r_{\max}$ increases. This observation also holds for q_2. Furthermore, a value of q_1 in Table 7 is less than the corresponding value in Table 8. The same observation holds for q_2. The value of q_2

is sometimes very high, although the absolute gap is actually rather small. This is due to the fact that the best known objective values (and the optimal values), used in the denominators, can be close to zero. This is an inherent property of some of the test problems that makes the *relative* gaps large.

We are not able to find such a pattern for q_3. It can be observed that the value of q_3 is most often between 20 and 40%. In Example 1 in Section 2.1, $\underline{v}^{\emptyset} = -113.5$, $\underline{v}^{I} = -110.01$ and $\bar{z}_P = v(P) = -54.0$. This gives $q_1 = 110.18\%$, $q_2 = 103.7\%$ and $q_3 = 5.88\%$, so about 5.88% of the original gap is closed when the constraints (8c) are appended.

In Tables 7 and 8, t_i is the time spent in the first inner repeat–until loop of *CGwLIoC*. The total CPU-time is denoted by t. Further, B is the portion of activated constraints of type (8c), as defined by (18). The values of B are consistently small. On average less than 1% of all the constraints of type (8c) are appended to the master program. As was discussed at the end of Section 5.2.4 and at the beginning of Section 3.4, the CPU-times increase considerably when these constraints are appended. The behavior of the column generation algorithm with late inclusion of constraints is very similar to the poor behavior reported on in [51], i.e., a large number of columns must be generated in order to reoptimize the new master program after appending a small set of constraints. A possible explanation is as follows.

Consider the column generation problem, *CGP*, which is described in Section 3.2. The objective coefficients in *CGP* are given by (11). It can be observed that for given edge costs and vertex revenues, not each pair $(x, y) \in F$ can be an optimal solution to *CGP*, since there might be no values of the multipliers α, β that makes this (x, y) optimal. Let Θ denote the number of distinct $(x, y) \in F$ that are optimal in *CGP* for some values of these multipliers. When some constraints of type (8c) are appended to the master program, the objective coefficients in *CGP* are given by (16). The multipliers μ introduce a new degree of freedom in the objective of *CGP*, and, as a result of this, more solutions $(x, y) \in F$ than previously are optimal in *CGP* for some values of the multipliers α, β and μ. Further, it seems reasonable to assume that the number of possible solutions to *CGP* grows quite fast with the number of constraints of type (8c) that are appended to the master program. (The behavior shown in Fig. 11 indicates that this is the case.)

In order to understand the consequences of this effect we note that column generation is equivalent to the solution of a Lagrangian dual problem by a cutting plane method. In our application, the column generation scheme is equivalent to a cutting plane method on the Lagrangian dual problem obtained when the vertex degree restrictions (1a) and the knapsack constraint (1c) in P_2 are Lagrangian relaxed. The resulting dual objective function is piecewise linear and the number of pieces equals Θ, since the Lagrangian relaxed problem coincide with *CGP*. The Lagrangian dual function has more segments when constraints of type (5) are present, and therefore is less nonsmooth. It will be harder to solve with a cutting plane methodology, compared to when no constraints of type (5) are present.

5.3.2. Numerical experiments using MCGPS

Above we have established the fact that the constraints (5) are useful for strengthening the linear programming relaxation of TSSP_3. In this subsection we investigate if the quality of the heuristic integer solutions obtained from *MCGPS*, see Section 4, is improved when constraints of type (8c) are appended to the master program. To do this, we execute the *MCGPS* heuristic in each of the 100 latest iterations of the procedure *CGwLIoC* before any constraints of type (8c) are included into the master program, and in each iteration of the entire procedure, respectively. Let $\bar{z}^1$ and $\bar{z}^2$ denote the best objective values found in these two phases, respectively. The values of $\bar{z}^1$ and $\bar{z}^2$ are compared to $v(MP^I)$ instead of $v(P)$, since the latter is unknown. Define $q_4 = [\bar{z}^1 - \bar{z}^2]/[\bar{z}^1 - v(MP^I)]$, which estimates how much closer $\bar{z}^2$ is to $v(P)$ than $\bar{z}^1$ is. In Tables 9 and 10, q_4 is computed as an average of five instances. The number $\bar{z}^2\_wins$ denotes the number of times that $\bar{z}^2$ is better (lower) than $\bar{z}^1$ out of these five instances.

It can be seen from Tables 9 and 10, that the constraints (8c), when appended to the master program, seem to improve the quality of the heuristic integer solutions on average. A general comment of the results, before computing the average, can be made. Whenever $\bar{z}^2$ is better (lower) than $\bar{z}^1$, then the difference between these values is often quite large. On the contrary, when $\bar{z}^1$ is better than $\bar{z}^2$, then the difference is typically quite small. This explains why all the average values of q_4 are positive.

6. Conclusions and further research

We proposed a decomposition of the traveling salesman subtour problem. This decomposition exploits a substructure that is similar to the well known 1-tree relaxation of the TSP [28]. Further, variable upper bound (VUB) constraints are used to strengthen the linear programming relaxation.

Table 9
Improvement of heuristic cycles when constraints of type (8c) are present in the master program, uncorrelated test problems

α	$\|V\|$	$r_{\max} = 40$		$r_{\max} = 100$	
		$\#\bar{z}^2\_wins$	q_4 (%)	$\#\bar{z}^2\_wins$	q_4 (%)
0.2	50	3	17.2	3	4.0
	100	4	25.9	5	30.4
0.5	50	5	32.2	3	4.9
	100	5	39.7	5	31.2
0.8	50	5	22.1	4	10.5
	100	5	34.0	5	23.4

Table 10
Improvement of heuristic cycles when constraints of type (8c) are present in the master program, correlated test problems

α	$\|V\|$	$r_{\max} = 40$		$r_{\max} = 100$	
		$\#\bar{z}^2\_wins$	q_4 (%)	$\#\bar{z}^2\_wins$	q_4 (%)
0.2	50	2	7.8	4	9.5
	100	5	23.4	4	18.3
0.5	50	5	18.9	4	13.7
	100	5	25.7	5	25.4
0.8	50	5	18.7	5	13.6
	100	5	24.0	4	14.1

The 1-tree structure of the new formulation is used in a stabilized column generation scheme. The stabilization technique is a slight modification of the boxstep method in [42], and involves an adaptive method for changing the size of the box during the computations. We experience large improvements of the computing times when well-calibrated boxes are used, compared to when no boxes are used.

It is further established that the linear programming relaxation is strengthened significantly by the addition of the VUB constraints. Moreover, only a small subset of these constraints are needed in order to obtain this effect. The addition of these relatively few constraints to the master program of the column generation scheme will, however, slow down the practical convergence rate of this scheme significantly. This behavior of a scheme that combines column generation with addition of new constraints is very similar to that reported on in [51], and it can probably be explained by the arguments at the end of Section 5.3.1. This troublesome behavior motivates further research in order to improve the performance of column generation schemes with delayed inclusion of some constraints. It might be worthwhile to study the effects of introducing box-constraints also in the master problem.

We have also designed a heuristic method for constructing feasible solutions to TSSP. This heuristic works by manipulating the solutions to the column generation problem with the aim of reaching overall feasibility. It is our experience from the computational results in this paper (and experiments that are not reported here) that this heuristic method often produces near-optimal solutions. We observe that the addition of VUB constraints to the master program typically leads to an improved quality of the heuristic solutions.

Concerning properties of TSSP with respect to the size of the duality gap, we note that the gap typically decreases whenever the size of the problem, the right hand side value of the knapsack constraint, or the level of the vertex revenues, increase. Moreover, the duality gaps for problems where the vertex weights are correlated to the vertex revenues, are larger than in the uncorrelated case.

An immediate opportunity for improving upon the proposed algorithm is to consider incorporating all constraints (1b), and not only those corresponding to $|S| = 2$, into the master problem. In order to identify violated constraints of this type, we would need to a invoke a non-trivial separation problem. (This separation amounts to solving $|V| - 1$ maximum flow problems.) The additional constraints thus generated would improve the quality of the lower bound. The amount

of computations would however increase, and the difficulties of combining column generation and late inclusions of constraints would probably become even more severe. The full utilization of the constraints (1b) is therefore a possible subject for future evaluation.

The solution technique presented in this paper can be adapted to vehicle routing problems; this is an interesting subject for future research. Another topic for future research is to attempt to accelerate a column generation scheme by finding a good, or even near-optimal, set of initial columns to the master program. This is of course a non-trivial task, and we are currently studying the use of subgradient optimization for this purpose.

Acknowledgments

This research has been financed by grants (98.16) from the Center for Industrial Information Technology, CENIIT, at Linköping university. The authors wish to thank Professor John Mittenthal for providing code for the Insert/Delete heuristic and Professor Marshall Fisher for providing the code for the solution of our column generation problem. We thank the two referees for thorough reviews and many valuable suggestions.

References

[1] L.H. Appelgren, Integer programming methods for a vessel scheduling problem, Transportation Sci. 5 (1971) 64–78.
[2] B. Awerbuch, Y. Azar, A. Blum, S. Vempala, New approximation guarantees for minimum-weight k-trees and prize-collecting salesman, SIAM J. Comput. 1 (1998) 254–262.
[3] E. Balas, The prize collecting traveling salesman problem, Networks 19 (1989) 621–636.
[4] E. Balas, The prize collecting traveling salesman problem: II. Polyhedral results, Networks 25 (1995) 199–216.
[5] E. Balas, G. Martin, ROLL-A-ROUND: Software package for scheduling the rounds of a rolling mill, Copyright Balas and Martin Associates, 104 Maple Heights Road, Pittsburgh, 1985.
[6] C. Barnhart, C.A. Hane, P.H. Vance, Using branch-and-price-and-cut to solve origin-destination multicommodity flow problems, Oper. Res. 48 (2000) 318–326.
[7] P. Bauer, The circuit polytope: facets, Math. Oper. Res. 22 (1997) 110–145.
[8] J.E. Beasley, An SST-based algorithm for the Steiner problem in graphs, Networks 19 (1989) 1–16.
[9] J.E. Beasley, E.M. Nascimento, The vehicle routing–allocation problem: a unifying framework, TOP 4 (1996) 65–86.
[10] D. Bienstock, M.X. Goemans, D. Simchi-Levi, D. Williamson, A note on the prize collecting traveling salesman problem, Math. Programming 59 (1993) 413–420.
[11] A.E. Bixby, C.R. Coullard, D. Simchi-Levi, The capacitated prize-collecting traveling salesman problem, Technical Report, Department of Industrial Engineering and Management Sciences, Northwestern University, 1997.
[12] M. Bowers, C.E. Noon, B. Thomas, A parallel implementation of the TSSP+1 decomposition for the capacity-constrained vehicle routing problem, Comput. Oper. Res. 23 (1996) 723–732.
[13] M. Dell'Amico, F. Maffioli, A. Sciomachen, A Lagrangian heuristic for the prize collecting travelling salesman problem, Ann. Oper. Res. 81 (1998) 289–305.
[14] M. Dell'Amico, F. Maffioli, P. Värbrand, On prize-collecting tours and the asymmetric travelling salesman problem, Internat. Trans. Oper. Res. 2 (1995) 297–308.
[15] O. du Merle, D. Villeneuve, J. Desrosiers, P. Hansen, Stabilized column generation, Discrete Math. 194 (1999) 229–237.
[16] D. Feillet, P. Dejax, M. Gendreau, The selective traveling salesman problem and extensions: an overview, Technical Report CER 00-05 A, Laboratoire Production Logistique, Ecole Centrale Paris, 2000.
[17] D. Feillet, P. Dejax, M. Gendreau, Traveling salesman problems with profits, Transportation Sci. 39 (2005) 188–205.
[18] M. Fischetti, J. Salazar, P. Toth, Solving the orienteering problem through branch-and-cut, INFORMS J. Comput. 10 (1998) 133–148.
[19] M. Fischetti, P. Toth, An additive approach for the optimal solution of the prize-collection travelling salesman problem, in: B.L. Golden, A.A. Assad (Eds.), Vehicle Routing: Methods and Studies, Elsevier Science Publishers B.V., North-Holland, Amsterdam, 1988.
[20] M. Fisher, A polynomial algorithm for the degree-constrained minimum k-tree problem, Oper. Res. 42 (1994) 775–779.
[21] M. Gendreau, G. Laporte, F. Semet, A branch-and-cut algorithm for the undirected selective traveling salesman problem, Networks 32 (1998) 263–273.
[22] D.H. Gensch, An industrial application of the traveling salesman's subtour problem, AIIE Trans. 10 (1978) 362–370.
[23] M.X. Goemans, D.P. Williamson, A general approximation technique for constrained forest problems, in: Proceedings from the Third SODA Symposium, 1992, pp. 307–316, (Chapter 37).
[24] B.L. Golden, L. Levy, R. Vohra, The orienteering problem, Naval Res. Logist. 34 (1987) 307–318.
[25] B.L. Golden, Q. Wang, L. Liu, A multifaceted heuristic for the orienteering problem, Naval Res. Logist. 35 (1988) 359–366.
[26] M. Göthe-Lundgren, K. Jörnsten, P. Värbrand, On the nucleolus of the basic vehicle routing game, Math. Programming 72 (1996) 83–100.
[27] M. Göthe-Lundgren, F. Maffioli, P. Värbrand, A Lagrangian decomposition approach for a prize collecting traveling salesman type problem, LiTH-MAT-R-1995-10, Linköping, Department of Mathematics, Linköping Institute of Technology, Linköping, Sweden, 1995.
[28] M. Held, R.M. Karp, The traveling-salesman problem and minimum spanning trees, Oper. Res. 18 (1970) 1138–1162.

[29] J.-B. Hiriart-Urruty, C. Lemaréchal, Convex Analysis and Minimization Algorithms II: Advanced Theory and Bundle Methods, Springer, Berlin, Germany, 1993.
[30] B. Jaumard, C. Meyer, T. Vovor, Column/row generation and elimination methods, Technical Report G-99-34, Department de génie électrique et de génic informatique, GERAD and École Polytechnique de Montréal, 1999.
[31] B. Kallehauge, J. Larsen, O.B.G. Madsen, Lagrangean duality applied on vehicle routing with time windows-experimental results, Technical Report IMM-TR-2001-9, Informatics and mathematical modelling, Technical University of Denmark, 2001.
[32] S. Kataoka, S. Morito, An algorithm for single constraint maximum collection problem, J. Oper. Res. 31 (1988) 515–530.
[33] S. Kataoka, T. Yamada, S. Morito, Minimum directed 1-subtree relaxation for score orienteering problem, European J. Oper. Res. 104 (1998) 139–153.
[34] M. Kubo, H. Kasugai, On symmetric subtour problems, Japan J. Ind. Appl. Math. 9 (1992) 383–396.
[35] M. Labbé, G. Laporte, I.R. Martin, J.J. Salazar, The median cycle problem, Technical Report, 1999, Available on ⟨http://citeseer.nj.nec.com/462934.html⟩.
[36] G. Laporte, Generalized subtour elimination constraints and connectivity constraints, J. Oper. Res. Soc. 37 (1986) 509–514.
[37] G. Laporte, S. Martello, The selective traveling salesman problem, Discrete Appl. Math. 26 (1990) 193–207.
[38] T. Larsson, M. Patriksson, C. Rydergren, A column generation procedure for the side constrained traffic equilibrium problem, Transportation Res. Part B 38 (2004) 17–38.
[39] L.S. Lasdon, Optimization Theory for Large Systems, Macmillan, New York, 1970.
[40] A.C. Leifer, M.B. Rosenwein, Strong linear programming relaxations for the orienteering problem, European J. Oper. Res. 73 (1994) 517–523.
[41] S. Lin, B.W. Kernighan, An effective heuristic algorithm for the traveling salesman problem, Oper. Res. 21 (1973) 498–516.
[42] R.E. Marsten, W.W. Hogan, J.W. Blackenship, The boxstep method for large-scale optimization, Oper. Res. 23 (1975) 389–405.
[43] J. Mittenthal, C.E. Noon, An insert/delete heuristic for the travelling salesman subset-tour problem with one additional constraint, J. Oper. Res. Soc. 43 (1992) 277–283.
[44] G.L. Nemhauser, S. Park, A polyhedral approach to edge coloring, Oper. Res. Lett. 10 (1991) 315–322.
[45] C.E. Noon, J. Mittenthal, R. Pillai, A TSSP+1 decomposition strategy for the vehicle routing problem, European J. Oper. Res. 79 (1994) 524–536.
[46] M. Padberg, G. Rinaldi, A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems, SIAM Rev. 33 (1991) 60–100.
[47] R. Pillai, The Traveling Salesman Subset-Tour Problem with one Additional Constraint (TSSP+1), Ph.D. Thesis, University of Tennessee, Knoxville, 1992.
[48] R. Ramesh, K. Brown, An efficient four-phase heuristic for the generalized orienteering problem, Comput. Oper. Res. 18 (1991) 151–165.
[49] R. Ramesh, Y. Yoon, M. Karwan, An optimal algorithm for the orienteering tour problem, ORSA J. Comput. 4 (1992) 155–165.
[50] T. Tsiligirides, Heuristic methods applied to orienteering, J. Oper. Res. Soc. 35 (1984) 797–809.
[51] J.M. van den Akker, C.A.J. Hurkens, M.W.P. Savelsbergh, Time-index formulations for machine scheduling problems: column generation, INFORMS J. Comput. 12 (2000) 111–124.
[52] F. Vanderbeck, L. Wolsey, An exact algorithm for IP column generation, Oper. Res. Lett. 19 (1996) 151–159.
[53] B. Verweij, K. Aardal, The merchant subtour problem, Math. Programming 94 (2003) 295–322.
[54] T. Volgenant, R. Jonker, On some generalizations of the travelling-salesman problem, J. Oper. Res. Soc. 38 (1987) 1073–1079.
[55] A. Westerlund, M. Göthe-Lundgren, T. Larsson, A note on relatives to the Held and Karp 1-tree problem, Oper. Res. Lett. 34 (2006) 275–282.
[56] W.E. Wilhelm, A technical review of column generation in integer programming, Optim. Eng. 2 (2001) 159–200.